# DISTRIBUTED DATA MANAGEMENT: UNSOLVED PROBLEMS AND NEW ISSUES[*]

M. Tamer Özsu[†]
GTE Laboratories Incorporated
40 Sylvan Road
Waltham, MA 02254
mto@gte.com

Patrick Valduriez
INRIA, Rocquencourt
78153 Le Chesnay
France
patrickv@madonna.inria.fr

## ABSTRACT

Distributed database technology is expected to have a significant impact on data processing in the upcoming years. With the introduction of commercial products, expectations are that distributed database management systems will by and large replace centralized ones within the next decade. In this paper, we reflect on the promises of distributed database technology, take stock of where we are, and discuss the issues that remain to be solved. We also present new research issues, such as distributed object-oriented systems, distributed knowledge bases, multiprocessor data servers, and distributed multidatabase systems, that arise with the introduction of new technology and the subsequent relaxation of some of the assumptions underlying current systems.

**Keywords:** distributed database, data distribution, transaction management, distributed query procesing, object-oriented system, knowledge base, multidatabase system.

[*] An abridged version of this paper has appeared as “Distributed Database Systems: Where Are We Now?", *Computer*, Vol. 24, No. 8, August 1991.

[†] On leave from Department of Computing Science, University of Alberta, Edmonton, Canada T6G 2H1; email: ozsu@cs.ualberta.ca.

## 1. INTRODUCTION

Distributed database technology is one of the more important developments of the past decade. During this period, distributed database research issues have been topics of intense study, culminating in the release of a number of “first generation” commercial products. Distributed database technology is expected to impact data processing the same way that centralized systems did a decade ago. It has been claimed that within the next ten years, centralized database managers will be an “antique curiosity” and most organizations will move toward distributed database managers [STON88, page 189].

The technology is now at the critical stage of finding its way into commercial products. At this juncture, it is important to seek answers to the following questions:

1. What were the initial goals and promises of the distributed database technology? How do the current commercial products measure up to these promises? In retrospect, were these goals achievable?

2. Have the important technical problems already been solved?

3. What are the technological changes that underlie distributed data managers and how will they impact next generation systems?

The last two questions hold particular importance for researchers since their answers lay down the road map for research in the upcoming years. Recent papers that address these questions have emphasized scaling problems [STON89] and issues related to the introduction of heterogeneity and autonomy [GARC90a]. While these problems are important ones to address, there are many others that remain unsolved. Even the much studied topics such as distributed query processing and transaction management have research problems that have yet to be addressed adequately. Furthermore, new issues arise with the changing technology, expanding application areas, and the experience that has been gained with the limited application of the distributed database technology. The most important new topics that can be identified for future research are distributed object-oriented database systems and distributed knowledge base systems. Additionally, database machines which were hot research topics in the 1970's but had lost some of their lustre in the 1980's [BORA83] are likely to reemerge as multiprocessor-based parallel data servers on distributed systems [DEWI90a]. Finally, distributed heterogeneous database systems are now being revisited as their autonomy considerations are being recognized as more important than their heterogeneity. This changing recognition has resulted in an emphasis on multidatabase systems as a platform for the development of interoperable information systems.

In this paper our purpose is to address the above questions and provide answers to them. Our emphasis is on answering these questions rather than providing a tutorial introduction to distributed database technology or a survey of the capabilities of existing products. We introduce each topic very briefly to establish terminology and then proceed to discuss the issues raised in the above questions. For more detailed presentation of the technology, we direct the user to the various textbooks on the subject.

The organization of this paper is as follows. In Section 2, we present a view of distributed database systems to serve as the framework for the rest of the paper. In Section 3, we present the potential advantages from the perspective of both end users and application designers and discuss the degree to which they have been met by the first generation distributed DBMSs. Section 4 discusses open research problems. Section 5 concentrates on the next generation distributed database systems and the related new research issues.

## 2. WHAT IS A DISTRIBUTED DATABASE SYSTEM?

A distributed database (DDB) is *a collection of multiple, logically interrelated databases distributed over a computer network* [ÖZSU91]. A distributed database management system (distributed DBMS) is then defined as *the software system that permits the management of the DDB and makes the distribution transparent to the users*. We use the term distributed database system (DDBS) to refer to the combination of the DDB and the distributed DBMS. Assumptions regarding the system that underlie these definitions are:

1. Data is stored at a number of sites. Each site is assumed to *logically* consist of a single processor. Even if some sites are multiprocessor machines, the distributed DBMS is not concerned with the storage and management of data on this parallel machine. We address the issues related to parallel database servers in Section 5.3.

2. The processors at these sites are interconnected by a computer network rather than a multiprocessor configuration. The important point here is the emphasis on loose-interconnection between processors which have their own operating systems and operate independently. Even though shared-nothing multiprocessor architectures are quite similar to the loosely interconnected distributed systems, they have different issues to deal with (e.g., task allocation and migration, load balancing, etc.) that are not considered in this paper.

3. The DDB is a database, not some "collection" of files that can be individually stored at each node of a computer network. This is the distinction between a DDB and a collection of files managed by a distributed file system. To form a DDB, distributed data should be logically

related, where the relationship is defined according to some structural formalism, and access to data should be at a high level via a common interface. The typical formalism that is used for establishing the logical relationship is the relational model. In fact, most existing distributed database system research assumes a relational system.

4. The system has the full functionality of a DBMS. It is neither, as indicated above, a distributed file system, nor a transaction processing system [BERN90]. Transaction processing is not only one type of distributed application, but it is also among the functions provided by a distributed DBMS. However, a distributed DBMS provides other functions such as query processing, structured organization of data, and so on that transaction processing systems do not necessarily deal with.

These assumptions are valid in today's technology base. Most of the existing distributed systems are built on top of local area networks in which each site is usually a single computer. The database is distributed across these sites such that each site typically manages a single local database (Figure 1). This is the type of system that we concentrate on for the most part of this paper. However, next generation distributed DBMSs will be designed differently as a result of technological developments – especially the emergence of affordable multiprocessors and high-speed networks – the increasing use of database technology in application domains which are more complex than business data processing, and the wider adoption of client-server mode of computing accompanied by the standardization of the interface between the clients and the servers. Thus, the next generation distributed DBMS environment will include multiprocessor database servers connected to high speed networks which link them and other data repositories to client machines that run application code and participate in the execution of database requests [TAYL87]. Distributed relational DBMSs of this type are already appearing and a number of the existing object-oriented systems [KIM89, ZDON90] also fit this description. Throughout the paper we will indicate where the existing systems fall short of the desired functionality. The research problems related to next generation systems that await solution are the topics of Section 5. Until then, our emphasis will be on systems which follow the above assumptions.

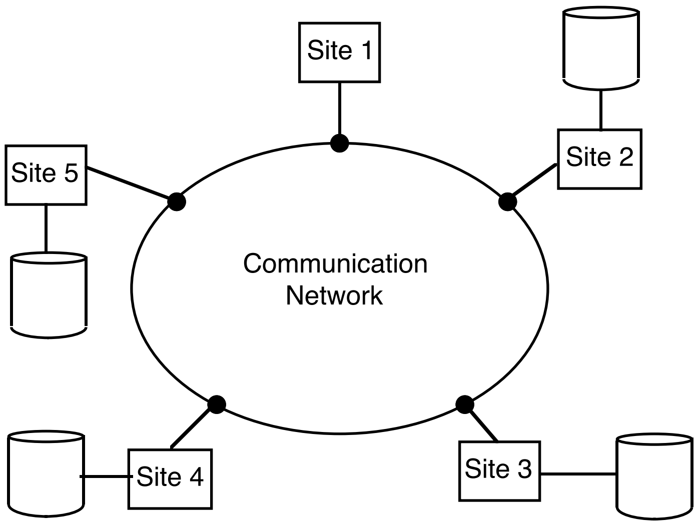

Figure 1. A Distributed Database Environment

A distributed DBMS as defined above is only one way of providing database management support for a distributed computing environment. In [ÖZSU91] we present a working classification of possible design alternatives along three dimensions: autonomy, distribution, and heterogeneity.

- *Autonomy* refers to the distribution of control, and indicates the degree to which individual DBMSs can operate independently. It involves a number of factors such as whether the component systems exchange information[1], whether they can independently execute transactions, and whether one is allowed to modify them. Three types of autonomy are tight integration, semiautonomy and full autonomy (or total isolation). In tightly integrated systems a single-image of the entire database is available to users who want to share the information which may reside in multiple databases. Semiautonomous systems consist of DBMSs that can (and usually do) operate independently, but have decided to participate in a federation to make their local data shareable. In totally isolated systems, however, the individual components are stand-alone DBMSs which know neither of the existence of other DBMSs nor of how to communicate with them.

- *Distribution* dimension of the taxonomy deals with data. We consider two cases, namely, either data are physically distributed over multiple sites that communicate with each other

[1] In this context "exchanging information" does not refer to networking concerns, but whether the DBMSs are designed to exchange information and coordinate their actions in executing user requests.

over some form of communication medium or they are stored at only one site.

- *Heterogeneity* can occur in various forms in distributed systems, ranging from hardware heterogeneity and differences in networking protocols to variations in data managers. The important ones from the perspective of database systems relate to data models, query languages, interfaces, and transaction management protocols. The taxonomy classifies DBMSs as homogeneous or heterogeneous.

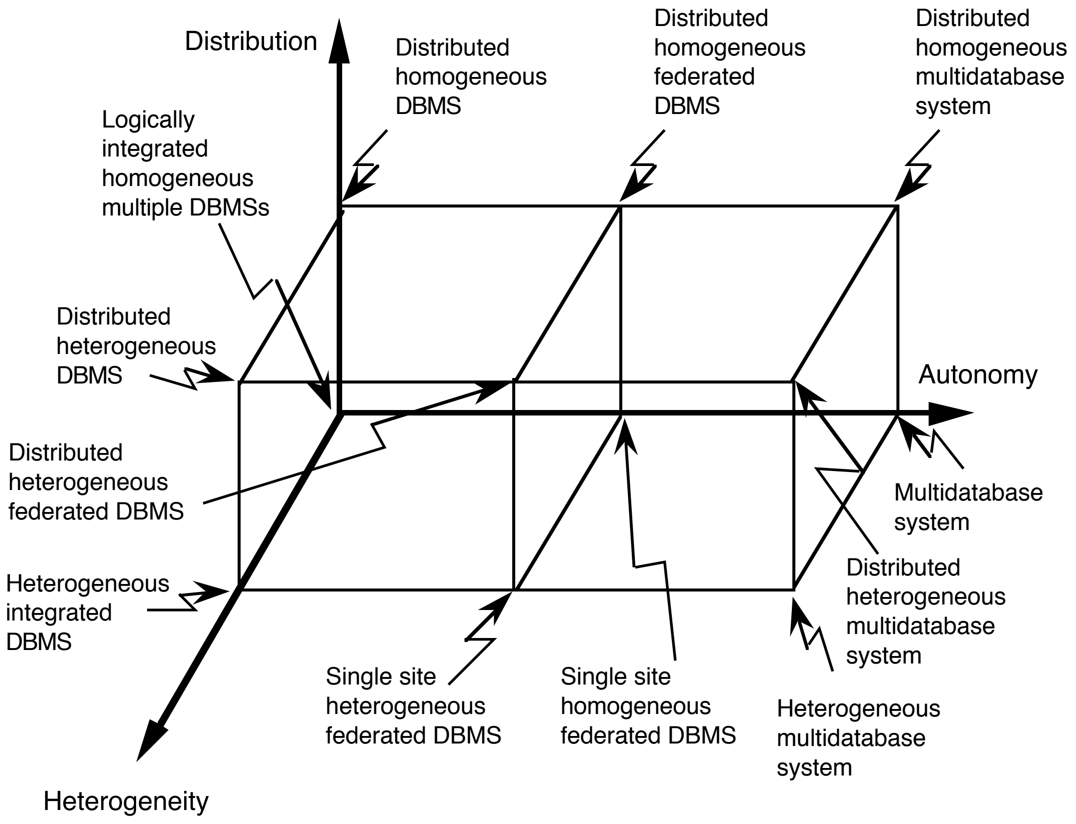
Distribution
Distributed homogeneous DBMS
Distributed homogeneous federated DBMS
Distributed homogeneous multidatabase system
Logically integrated homogeneous multiple DBMSs
Distributed heterogeneous DBMS
Autonomy
Distributed heterogeneous federated DBMS
Multidatabase system
Heterogeneous integrated DBMS
Distributed heterogeneous multidatabase system
Single site heterogeneous federated DBMS
Single site homogeneous federated DBMS
Heterogeneous multidatabase system
Heterogeneity

Figure 2. Implementation Alternatives

The alternative system architectures based on this taxonomy are illustrated in Figure 2. The arrows along the axes do not indicate an infinite number of choices, but simply the dimensions of the taxonomy that we discussed above. For most of this paper, we deal with tightly integrated, distributed, and homogeneous database systems. In Section 5.4, we consider distributed, multidatabase systems.

## 3. THE CURRENT STATE OF DISTRIBUTED DATABASE TECHNOLOGY

As with any emerging technology, DDBSs have their share of fulfilled and unfulfilled promises. In this section, we consider the commonly cited advantages of distributed DBMSs and discuss how well the existing commercial products provide these advantages.

### 3.1 Transparent Management of Distributed and Replicated Data

Centralized database systems have taken us from a paradigm of data processing, in which data definition and maintenance was embedded in each application, to one in which these functions are abstracted out of the applications and placed under the control of a server called the DBMS. This new orientation results in *data independence,* whereby the application programs are immune to changes in the logical or physical organization of the data and vice versa. The distributed database technology intends to extend the concept of data independence to environments where data is distributed and replicated over a number of machines connected by a network. This is provided by several forms of *transparency*: *network* (and, therefore, *distribution*) transparency, *replication* transparency, and *fragmentation* transparency. Transparent access to data separates the higher-level semantics of a system from lower-level implementation issues. Thus the database users would see a logically integrated, single image database even though it may be physically distributed, enabling them to access the distributed database as if it was a centralized one. In its ideal form, full transparency would imply a query language interface to the distributed DBMS which is no different from that of a centralized DBMS.

Most commercial distributed DBMSs do not provide a sufficient level of transparency. Part of this is due to the lack of support for the management of replicated data. A number of systems do not permit replication of the data across multiple databases while those that do require that the user be physically “logged on” to one database at a given time. Some distributed DBMSs attempt to establish their own transparent naming scheme, usually with unsatisfactory results, requiring the users either to specify the full path to data or to build aliases to avoid long path names. An important aspect of the problem is the lack of proper operating system support for transparency. Network transparency can easily be supported by means of a transparent naming mechanism that can be implemented by the operating system. The operating system can also assist with replication transparency, leaving the task of fragmentation transparency to the distributed DBMS. We discuss replication issues in Section 4.3 and the role of the operating system in Section 4.5.

Full transparency is not a universally accepted objective, however. Gray argues that full transparency makes the management of distributed data very difficult and claims that "applications coded with transparent access to geographically distributed databases have: poor manageability, poor modularity, and poor message performance" [GRAY89]. He proposes a remote procedure call mechanism between the requestor users and the server DBMSs whereby the users would direct their queries to a specific DBMS. We agree that the management of distributed data is more difficult if transparent access is provided to users, and that the client-server architecture with a remote procedure call-based communication between the clients and the servers is the right architectural approach. In fact, some commercial distributed DBMSs are organized in this fashion (e.g., Sybase). However, the original goal of distributed DBMSs to provide transparent access to distributed and replicated data should not be given up due to these difficulties. The issue is who should be taking over the responsibility of managing distributed and replicated data: the distributed DBMS or the user application? In our opinion, it should be the distributed DBMS whose components may be organized in a client-server fashion. The related technical issues are among the remaining research and development issues that need to be addressed.

### 3.2 Reliability Through Distributed Transactions

Distributed DBMSs are intended to improve reliability since they have replicated components and, thereby eliminate single points of failure. The failure of a single site, or the failure of a communication link which makes one or more sites unreachable, is not sufficient to bring down the entire system[2]. In the case of a distributed database, this means that some of the data may be unreachable, but with proper care, users may be permitted to access other parts of the distributed database. The "proper care" comes in the form of support for *distributed transactions*.

A *transaction* is a basic unit of consistent and reliable computing, consisting of a sequence of database operations executed as an atomic action. It transforms a consistent database state to another consistent database state even when a number of such transactions are executed concurrently (sometimes called *concurrency transparency*), and even when failures occur (also called *failure atomicity*). Therefore, a DBMS that provides full transaction support guarantees that concurrent execution of user transactions will not violate database consistency in the face of system failures as long as each transaction is correct, i.e., obeys the integrity rules specified on the database.

Distributed transactions execute at a number of sites at which they access the local database.

[2]We do not wish to discuss the differences between site failures and link failures at this point. It is well-known that link failures may cause network partitioning and are, therefore, more difficult to deal with.

With full support for distributed transactions, user applications can access a single logical image of the database and rely on the distributed DBMS to ensure that their requests will be executed correctly no matter what happens in the system. “Correctly” means that user applications do not need to be concerned with coordinating their accesses to individual local databases nor do they need to worry about the possibility of site or communication link failures during the execution of their transactions. This illustrates the link between distributed transactions and transparency, since both involve issues related to distributed naming and directory management, among other things.

Providing transaction support requires the implementation of distributed concurrency control and distributed reliability protocols, which are significantly more complicated than their centralized counterparts. The typical distributed concurrency control algorithm is some variation of the well-known two-phase locking (2PL) depending upon the placement of the lock tables and the assignment of the lock management responsibilities. Distributed reliability protocols consist of distributed commit protocols and recovery procedures. Commit protocols enforce atomicity of distributed transactions by ensuring that a given transaction has the same effect (commit or abort) at each site where it exists, whereas the recovery protocols specify how the global database consistency is to be restored following failures. In the distributed environment, the commit protocols are two-phase (2PC). In the first phase, an agreement is established among the various sites regarding the fate of a transaction. The agreed upon action is taken in the second phase.

Data replication increases database availability since copies of the data stored at a failed or unreachable site (due to link failure) exist at other operational sites. However, supporting replicas require the implementation of *replica control protocols* that enforce a specified semantics of accessing them. The most straightforward semantics is *one-copy equivalence* which can be enforced by the ROWA protocol (“read one write all”). In ROWA, a logical read operation on a replicated data item is converted to one physical read operation on any one of its copies, but a logical write operation is translated to physical writes on all of the copies. More complicated replica control protocols that are less restrictive and that are based on deferring the writes on some copies have been studied, but are not implemented in any of the systems that we know. More on this in Section 4.3.

Concurrency control and commit protocols are among the two most researched topics in distributed databases. Yet, their implementation in existing commercial systems is not widespread. The performance implications of implementing distributed transactions (which are not fully understood, as we discuss in Section 4.1) make them unpopular among vendors. Commercial systems provide varying degrees of distributed transaction support. Some (e.g., Oracle) require users to have one database open at a given time, thereby eliminating the need for distributed transactions while others (e.g., Sybase) implement the basic primitives that are necessary for the 2PC protocol, but require the user applications to handle the coordination of the commit actions. In other words, the distributed DBMS does not enforce atomicity of distributed transactions, but provide the basic primitives by which user applications can enforce it. There are other systems, however, that implement the 2PC protocols fully (e.g., Ingres and NonStop SQL).

## 3.3 Improved Performance

The case for the improved performance of distributed DBMSs is typically made based on two points. First, a distributed DBMS fragments the conceptual database, enabling data to be stored in close proximity to its points of use (also called *data localization*). This has two potential advantages: (1) since each site handles only a portion of the database, contention for CPU and I/O services is not as severe as for centralized databases, and (2) localization reduces remote access delays that are usually involved in wide area networks (for example, the minimum round-trip message propagation delay in satellite-based systems is about 1 second). Most distributed DBMSs are structured to gain maximum benefit from data localization. Full benefits of reduced contention and reduced communication overhead can be obtained only by a proper fragmentation and distribution of the database (discussed in Section 4.2).

Second, the inherent parallelism of distributed systems may be exploited for inter-query and intra-query parallelism. *Inter-query parallelism* results from the ability to execute multiple queries at the same time while *intra-query parallelism* is achieved by breaking up a single query into a number of subqueries each of which is executed at a different site, accessing a different part of the distributed database.

If the user access to the distributed database consisted only of querying (i.e., read-only access), then provision of inter-query and intra-query parallelism would imply that as much of the database as possible should be replicated. However, since most database accesses are not read-only, the mixing of read and update operations requires the implementation of elaborate concurrency control and commit protocols.

Existing commercial systems employ two alternative execution models (other than the implementation of full distributed transaction support) in realizing improved performance. The first alternative is to have the database open only for queries (i.e., read-only access) during the regular operating hours while the updates are batched. The database is then closed to query activity during off-hours when the batched updates are run sequentially. This is time multiplexing between read activity and update activity. A second alternative is based on multiplexing the database. Accordingly, two copies of the database are maintained, one for ad hoc querying (called the *query* *database*) and the other for updates by application programs (called the *production database*). At regular intervals, the production database is copied to the query database. The latter alternative does not eliminate the need to implement concurrency control and reliability protocols for the production database since these are necessary to synchronize the write operations on the same data; however, it improves the performance of the queries since they can be executed without the overhead of transaction manipulation.

The performance characteristics of distributed database systems are not very well understood. There are not a sufficient number of true distributed database applications to provide a sound base to make practical judgements. In addition, the performance models of distributed database systems are not sufficiently developed. The database community has developed a number of benchmarks to test the performance of transaction processing applications, but it is not clear whether they can be used to measure the performance of distributed transaction management. The performance of the commercial DBMS products, even with respect to these benchmarks, are generally not openly published. NonStop SQL is one product for which performance figures, as well as the experimental setup that is used in obtaining them, has been published [TAND88].

## 3.4 Easier and More Economical System Expansion

In a distributed environment, it should be easier to accommodate increasing database sizes. Major system overhauls are seldom necessary; expansion can usually be handled by adding processing and storage power to the system. We call this *database size scaling*, as opposed to *network scaling* discussed in Section 4.1. It may not be possible to obtain a linear increase in "power," since this also depends on the overhead of distribution. However, significant improvements are still possible.

Microprocessor and workstation technologies have played a role in improving economies. The price/performance characteristics of these systems make it more economical to put together a system of smaller computers with the equivalent power of a single big machine. Many commercial distributed DBMSs operate on minicomputers and workstations to take advantage of their favorable price/performance characteristics. The current reliance on workstation technology is because most of the commercial distributed DBMSs operate within local area networks for which the workstation technology is most suitable. The emergence of distributed DBMSs that run on wide-area networks may increase the importance of mainframes. On the other hand, future distributed DBMSs may support hierarchical organizations where sites consist of clusters of computers communicating over a local area network with a high-speed backbone wide area network connecting the clusters. Whatever the architectural model, distributed DBMSs, with proper support for transparent access to data, will assist the incremental growth of database sizes without major impact on application code.

Another economic factor is the trade-off between data communication and telecommunication costs. In the previous section, we argued that data localization improves performance by reducing delays. It also reduces costs. Consider an application (such as inventory control) that needs to run at a number of locations. If this application accesses the database frequently, it may be more economical to distribute the data and to process it locally. This is in contrast to the execution of the application at various sites and making remote accesses to a central database that is stored at another site. In other words, the cost of distributing data and shipping some of it from one site to the other from time to time to execute distributed queries may be lower than the telecommunication cost of frequently accessing a remote database. We should state that this part of the economics argument is still speculative. As we indicated above, most of the distributed DBMSs are local area network products, and how they can be extended to operate in wide area networks is a topic of discussion and controversy (see Section 4.1).

## 4. UNSOLVED PROBLEMS

In the previous section we discussed the current state of commercial distributed DBMSs and how well they meet the original objectives that were set for the technology. Obviously, there is still some way to go before the commercial state-of-the-art fulfills the original goals of the technology. The issue is not only that the commercial systems have to catch up and implement the research results, but that there are still significant research problems that remain to be solved. The purpose of this section is to discuss these issues in some detail.

### 4.1 Network Scaling Problems

As noted before, the database community does not have a full understanding of the performance implications of all the design alternatives that accompany the development of distributed DBMSs. Specifically, questions have been raised about the scalability of some protocols and algorithms as the systems become geographically distributed [STON89] or as the number of system components increase [GARC90a]. Of specific concern is the suitability of the distributed transaction processing mechanisms (i.e., two-phase locking and, particularly, two-phase commit protocols) in wide area network-based distributed database systems. As we mentioned in Sections 3.2 and 3.3, there is a significant overhead associated with these protocols and implementing them over a slow wide area network may face difficulties [STON89].

Scaling issues are only one part of a more general problem, namely that we don't have a good handle on the role of the network architectures and protocols in the performance of distributed DBMSs. Almost all the performance studies that we know assume a very simple network cost model, sometimes as unrealistic as using a fixed communication delay that is independent of all network characteristics such as the load, message size, network size and so on. The inappropriateness of these models can be demonstrated easily. Consider, for example, a distributed DBMS that runs on an Ethernet-type local area network. Message delays in Ethernet increase as the network load increases, and, in general, cannot be bounded. Therefore, realistic performance models of an Ethernet-based distributed DBMS cannot realistically use a constant network delay or even a delay function which does not consider network load. In general, the performance of the proposed algorithm and protocols in different local area network architectures is not well understood, let alone their comparative behavior in moving from local area networks to wide area networks.

The proper way to deal with scalability issues is to develop general and sufficiently powerful performance models, measurement tools and methodologies. Such work for centralized DBMSs has been going on for some time, but has not yet been sufficiently extended to distributed DBMSs. We already raised questions about the suitability of the existing benchmarks. Detailed and comprehensive simulation studies do not exist either. Even though there are plenty of performance studies of distributed DBMS, these usually employ simplistic models, artificial workloads, conflicting assumptions or consider only a few special algorithms. It has been suggested that to make generalizations based on the existing performance studies requires a giant leap of faith. This does not mean that we do not have some understanding of the trade-offs. In fact, certain trade-offs have long been recognized and even the earlier systems have considered them in their design. For example, the query processor of the SDD-1 system [BERN81] was designed to execute distributed operations most efficiently on slow wide area networks. Later studies (e.g., [WAH85, ÖZSO87]) considered the optimization of query processors in faster, broadcasting local area networks. However, these trade-offs can mostly be spelled out only in qualitative terms; their quantification requires more research on performance models.

## 4.2 Distribution Design

The design methodology of distributed databases varies according to the system architecture. In the case of tightly integrated distributed databases, design proceeds top-down from requirements analysis and logical design of the *global database* to physical design of each *local database*. In the case of distributed multidatabase systems, the design process is bottom-up and involves the integration of existing databases. In this section we concentrate on the top-down design process issues.

The step in the top-down process that is of interest to us is *distribution design*. This step deals with designing the *local conceptual schemas* by distributing the global entities over the sites of the distributed system. The global entities are specified within the *global conceptual schema*. In case of the relational model, both the global and the local entities are relations and distribution design maps global relations to local ones. The most important research issue that requires attention is the development of a practical distribution design methodology and its integration into the general data modeling process.

There are two aspects of distribution design: *fragmentation* and *allocation*. Fragmentation deals with the partitioning of each global relation into a set of fragment relations while allocation concentrates on the (possibly replicated) distribution of these local relations across the sites of the distributed system. Research on fragmentation has concentrated on horizontal (i.e., selecting) and vertical (i.e., projecting) fragmentation of global relations. Numerous allocation algorithms based on mathematical optimization formulations have also been proposed.

There is no underlying design methodology that combines the horizontal and vertical partitioning techniques; horizontal and vertical partitioning algorithms have been developed completely independently. If one starts with a global relation, there are algorithms to decompose it horizontally as well as algorithms to decompose it vertically into a set of fragment relations. However, there are no algorithms that fragment a global relation into a set of fragment relations some of which are decomposed horizontally and others vertically. It is always pointed out that most real-life fragmentations would be mixed, i.e., would involve both horizontal and vertical partitioning of a relation, but the methodology research to accomplish this is lacking. What is needed is a distribution design methodology which encompasses the horizontal and vertical fragmentation algorithms and uses them as part of a more general strategy. Such a methodology should take a global relation together with a set of design criteria and come up with a set of fragments some of which are obtained via horizontal and others obtained via vertical fragmentation.

The second part of distribution design is allocation which is typically treated independently of fragmentation. The process is, therefore, linear when the output of fragmentation is input to allocation. At first sight, the isolation of the fragmentation and the allocation steps appears to simplify the formulation of the problem by reducing the decision space. However, closer examination reveals that isolating the two steps actually contributes to the complexity of the allocation models. Both steps have similar inputs, differing only in that fragmentation works on global relations whereas allocation considers fragment relations. They both require information about the user applications (e.g., how often they access data, what the relationship of individual data objects to one another is, etc.), but ignore how each other makes use of these inputs. The end result is that the fragmentation algorithms decide how to partition a relation based partially on how applications access it, but the allocation models ignore the part that this input plays in fragmentation. Therefore, the allocation models have to include all over again detailed specification of the relationship among the fragment relations and how user applications access them. What would be more promising is to extend the methodology discussed above so that the interdependence of the fragmentation and the allocation decisions is properly reflected. This requires extensions to existing distribution design strategies (e.g., [CERI87]).

We recognize that integrated methodologies such as the one we propose here may be considerably complex. However, there may be synergistic effects of combining these two steps enabling the development of quite acceptable heuristic solution methods. There are some studies that give us hope that such integrated methodologies and proper solution mechanisms can be developed (e.g., [MURO85, YOSH85]). These methodologies build a simulation model of the distributed DBMS, taking as input a specific database design, and measure its effectiveness. Development of tools based on such methodologies, which aid the human designer rather than attempt to replace him, is probably the more appropriate approach to the design problem.

## 4.3 Distributed Query Processing

Distributed query processors automatically translate a high-level query on a distributed database, which is seen as a single central database by the users, into an efficient low-level execution plan expressed on the local databases. Such translation has two important aspects. First, the translation must be a correct transformation of the input query so that the execution plan actually produces the expected result. The formal basis for this task is the equivalence between relational calculus and relational algebra, and the transformation rules associated with relational algebra. Second, the execution plan must be "optimal," i.e., it must minimize a cost function that captures resource consumption. This requires investigating equivalent alternative plans in order to select the best one.

Because of the difficulty of addressing these two aspects together, they are typically isolated in two sequential steps which we call *data localization* and *global optimization* in [ÖZSU91]. These steps are generally preceded by query decomposition which simplifies the input query and rewrites it in relational algebra. Data localization transforms an input algebraic query expressed on the distributed database into an equivalent fragment query (i.e., a query expressed on database fragments stored at different sites) which can be further simplified by algebraic transformations. Global query optimization generates an optimal execution plan for the input fragment query by making decisions regarding operation ordering, data movement between sites and the choice of both distributed and local algorithms for database operations. There are a number of problems regarding this last step. They have to do with the restrictions imposed on the cost model, the focus on a subset of the query language, the trade-off between optimization cost and execution cost, and the optimization/reoptimization interval.

The cost model is central to global query optimization since it provides the necessary abstraction of the distributed DBMS execution system in terms of access methods, and an abstraction of the database in terms of physical schema information and related statistics. The cost model is used to predict the execution cost of alternative execution plans for a query. A number of important restrictions are often associated with the cost model, limiting the effectiveness of optimization. It is a weighted combination of cost components such as I/O, CPU and communication and can capture either response time (RT) or total time (TT). However, the performance objective of many distributed systems is to improve throughput. The open problem here is to factor out the knowledge of the workload of concurrent queries within the cost model. Although TT optimization may increase throughput by minimizing resource consumption, RT optimization may well hurt throughput because of the overhead of parallelism. A potentially beneficial direction of research is to apply multiple query optimization [SELL88] where a set of important queries from the same workload are optimized together. This would provide opportunities for load balancing and for exploiting common intermediate results. Other problems associated with the cost model are the accuracy of the cost functions for distributed algorithms and the impact of update queries on throughput, particularly in the case of replicated databases. Careful analysis of the cost functions should provide insights for determining useful heuristics to cut down the number of alternative execution plans (see, for example, [ÖZSU90a, ÖZSU90b]). Work in extensible query optimization [FREY87] can be useful in parameterizing the cost model which can then be refined after much experimentation.

Even though query languages are becoming increasingly powerful (e.g., new versions of SQL), global query optimization typically focuses on a subset of the query language, namely select-project-join (SPJ) queries with conjunctive predicates. This is an important class of queries for which good optimization opportunities exist. As a result, a good deal of theory has been developed for join and semijoin ordering. However, there are other important queries that warrant optimization, such as queries with disjunctions, unions, aggregations or sorting. Although this has been partially addressed in centralized query optimization, the solutions cannot always be extended to distributed execution. For example, transforming an average function over a relation into multiple average functions, each defined on a fragment of the original relation, simply does not produce the correct result. One way of dealing with these queries is to centralize the operands (i.e., reconstruct the original relations) before processing the query, but this is clearly not desirable since it eliminates intra-query parallelism. Methods have to be found for processing complex queries without giving up parallelism. One can then have multiple optimization “experts,” each specialized in one class of queries (e.g., SPJ optimization, aggregate optimization, etc). As query languages evolve towards more general purpose languages which permit multiple statements to be combined with control statement (e.g., Sybase's TRANSACT-SQL) the major problem becomes the recognition of optimizable constructs within the input query. A promising solution is to separate the language understanding from the optimization itself which can be dedicated to several optimization experts [VALD89].

There is a necessary trade-off between *optimization cost* and *quality* of the generated execution plans. Higher optimization costs are probably acceptable to produce “better” plans for repetitive queries, since this would reduce query *execution cost* and amortize the optimization cost over many executions. However, it is unacceptable for ad hoc queries which are executed only once. The optimization cost is mainly incurred by searching the solution space for alternative execution plans. Thus, there are two important components that affect optimization cost: the size of the solution space and the efficiency of the search strategy. In a distributed system, the solution space can be quite large because of the wide range of distributed execution strategies. Therefore, it is critical to study the application of efficient search strategies [IOAN90] that avoid the exhaustive search approach. More important, a different search strategy should be used depending on the kind of query (simple vs. complex) and the application requirements (ad hoc vs. repetitive). This requires support for controllable search strategies [LANZ90].

Global query optimization is typically performed prior to the execution of the query, hence called static, for two reasons. First, it can be done within a compiler, reducing optimization cost. Second, it can better exploit knowledge regarding physical schema and data placement. Earlier proposals have proposed dynamic techniques which mix optimization and execution. This can be achieved easily using the static approach by adding tests at compile-time in order to make run-time decisions. A major problem with this approach is that the cost model used for optimization may become inaccurate because of changes in the fragment sizes or database reorganization which is important for load balancing. The problem, therefore, is to determine the optimal intervals of recompilation/reoptimization of the queries taking into account the trade-off between optimization and execution cost.

## 4.4 Distributed Transaction Processing

It may be hard to believe that in an area as widely researched as distributed transaction processing there may still be important topics to investigate, but there are. We have already mentioned in

Section 4.1 the scaling problems of transaction management algorithms. Additionally replica control protocols, more sophisticated transaction models, and non-serializable correctness criteria require further attention.

In replicated distributed DBMSs, database operations are specified on logical data objects[3]. The replica control protocols are responsible for mapping an operation on a logical data object to an operation on multiple physical copies of this data object. In so doing, they ensure the *mutual consistency* of the replicated database. The ROWA rule that we discussed in Section 3.2 is the most straightforward method of enforcing mutual consistency. Accordingly, a replicated database is in a mutually consistent state if all the copies of every data object have identical values.

The field of data replication needs further experimentation, research on replication methods for computation and communication, and more work to enable the systematic exploitation of application-specific properties. The field ripened for methodical experimental research only in the late 1980's, consequently, much still needs to be done. It remains difficult to evaluate the claims that are made by algorithm and system designers, and we lack a consistent framework for comparing competing techniques. One of the difficulties in quantitatively evaluating replication techniques lies in the absence of commonly accepted *failure incidence models*. For example, Markov models that are sometimes used to analyze the availability achieved by replication protocols assume the statistical independence of individual failure events, and the rarity of network partitions relative to site failures. We do not currently know that either of these assumptions is tenable, nor do we know how sensitive Markov models are to these assumptions. The validation of the Markov models by simulation cannot be trusted in the absence of empirical measurements, since simulations often embody the same assumptions that underlie the Markov analysis. Thus, there is a need for empirical studies to monitor failure patterns in real-life production systems, with the purpose of constructing a simple model of typical *failure loads*.

A failure load model should account for sequences of failures which raises some interesting research questions. If failures are not statistically independent, must the models employ joint probability distributions, or do correlation coefficients suffice? How inaccurate are the models that make the independence assumption? A failure model should include a variety of what might be called *failure incidence benchmarks* and availability metrics that are readily measurable. The development of commonly accepted failure models will clarify how reconfiguration algorithms should exploit information about failures that have been detected and diagnosed. Some methods (e.g., primary copy [ALSB76], available copies [BERN84]) call for immediate reorganization after a

[3]We use the term "data object" here instead of the more common "data item" because we do not want to make a statement about the granularity of the logical data.

failure before any further operations on the replicated data can be processed. On the other hand, voting methods [THOM79, GIFF79, HERL86] call for a reorganization only if the availability must (and can) be enhanced in the face of further failures that may occur before the first one is repaired.

Until failure loads are clarified and documented, validating the results obtained by analytical and simulation approaches, the most credible alternative is to construct testbed systems for the purpose of evaluating the relative availability and performance achieved by different replication techniques. These empirical studies must necessarily involve the tight integration of a number of different algorithms to maintain consistency while achieving replication, reconfiguration, caching, garbage collection, concurrent operation, replica failure recovery and security.This high degree of integration is necessitated by the tight and often subtle interaction between many of the different algorithms that form the components of the replication suite of algorithms (e.g., [WEIH89]).

The second area of research that needs work, is that of replicating computation and communication (including input and output). To achieve the twin goals of data replication, namely *availability* and *performance* (also see Sections 3.2 and 3.3), we need to provide integrated systems in which the replication of data goes hand in hand with the replication of computation and communication (including I/O). Only data replication has been studied intensely; relatively little has been done in the replication of computation and communication. Replication of computation has been studied for a variety of purposes, including running synchronous duplicate processes as “hot standbys” [GRAY85, COOP85], and processes implementing different versions of the same software to guard against human design errors [AVIZ85]. Replication of communication messages primarily by retry has been studied in the context of providing reliable message delivery [BIRM87], and a few papers report on the replication of input/output messages to enhance the availability of transactional systems [MINO82]. However, more work needs to be done to study how these tools may be integrated together with data replication to support such applications as real time control systems, that may benefit from all three kinds of replication. This work would be invaluable in guiding operating system and programming language designers towards the proper set of tools to offer in support of fault-tolerant systems.

In addition to replication, but related to it, work is required on more elaborate transaction models, especially those that exploit the semantics of the application. Higher availability and performance, as well as concurrency, can be achieved this way. As database technology enters new application domains such as engineering design, software development and office information systems, the nature and requirements for transactions change. Thus, work is needed on more complicated transaction models and on correctness conditions different from serializability.

As a first approximation, the existing work on transaction models can be classified along two dimensions: the transaction model and the structure of objects that they operate on. Along the transaction model dimension, we recognize flat transactions, closed nested transactions [MOSS85] and open transaction models such as sagas [GARC87], and models that include both open and closed nesting [BUCH91; ELMA90a], in increasing order of complexity. Along the object structure dimension, we identify simple objects (e.g., pages), objects as instances of abstract data types (ADTs), and complex objects, again in increasing complexity. We make the distinction between the last two to indicate that objects as instances of abstract data types support encapsulation (and therefore are harder to run transactions on than simple objects), but do not have a complex structure (i.e., do not contain other objects) and their types do not participate in an inheritance hierarchy. As stated above, this classification is not meant to be taken as a definitive statement, but only to highlight some of the transaction management issues. It is described in more detail in [BUCH91].

Within the above framework, most of the transaction model work in distributed systems has concentrated on the execution of flat transactions on simple objects. This point in the design space is quite well understood. While some work has been done in the application of nested transactions on simple objects, much remains to be done, especially in distributed databases. Specifically, the semantics of these transaction models are still being worked out [BEER87, LYNC86]. Similarly, there has been work done on applying simple transactions to objects as instances of abstract data types [WEIH88] and to complex objects [BADR88]. Again, these are initial attempts which need to be followed up to specify their full semantics, their incorporation into a DBMS, their interaction with recovery managers and, finally, their distribution properties.

Complex transaction models are important in distributed systems for a number of reasons. First, as we discuss in Section 5.4, transaction processing in distributed multidatabase systems can benefit from the relaxed semantics of these models. Second, the new application domains that distributed DBMSs will support in the future (e.g., engineering design, office information systems, cooperative work, etc.) require transaction models that incorporate more abstract operations that execute on complex data. Furthermore, these applications have a different sharing paradigm than the typical database access that we are accustomed to. For example, computer-assisted cooperative work environments [GRIE88] require participants to cooperate in accessing shared resources rather than competing for them as is usual in typical database applications. These changing requirements necessitate the development of new transaction models and accompanying correctness criteria.

A related line of research is multilevel transactions [BEER88] which view a transaction as consisting of multiple levels of abstract operations rather than a flat sequence of atomic actions. In this model, an atomic abstract operation at one level of abstraction is executed as an atomic transaction at the next lower level of abstraction. As we point out in Section 4.5, this approach may be beneficial in implementing transactions as operating system primitives.

## 4.5 Integration with Distributed Operating Systems

The undesirability of running a centralized or distributed DBMS as an ordinary user application on top of a host operating system (OS) has long been recognized [GRAY79, STON81, CHRI87]. There is a mismatch between the requirements of the DBMS and the functionality of the existing OSs. This is even more true in the case of distributed DBMSs which require functions (e.g., distributed transaction support including concurrency control and recovery, efficient management of distributed persistent data, more complicated access methods) that existing distributed OSs do not provide. Furthermore, distributed DBMSs necessitate modifications in how the distributed OSs perform their traditional functions (e.g., task scheduling, naming, buffer management). The architectural paradigms that guide the development of many commercial OSs do not permit easy incorporation of these considerations in OS designs. The result is either the modification of the underlying OS services by DBMS vendors, or the duplication of these services within a DBMS kernel which bypasses the OS. There is a need for a comprehensive treatment of these issues in the distributed setting.

A proper discussion of distributed DBMS/distributed OS integration issues can easily be the topic of a paper about the size of this one. We, therefore, only highlight the fundamental issues in this section. They relate to the appropriate system architecture, transparent naming of resources, persistent data management, distributed scheduling, remote communication and transaction support.

An important architectural consideration is that the coupling of distributed DBMSs and distributed OSs is not a binary integration issue. There is also the communication network protocols that need to be considered, adding to the complexity of the problem. Typically, distributed OSs implement their own communication services, ignoring the standards developed for the services offered by the communication networks. However, if OSs diverge significantly from the standard network protocols, porting the operating system to different networks can require major re-work to accommodate the idiosyncrasies of each individual network architecture. Interfacing the operating system with a network standard at a reasonable level is essential for portability and compatibility with the developments in networking standards. Thus the architectural paradigm has to be flexible enough to accommodate distributed DBMS functions, distributed operating system services as well as the communication protocol standards such as the ISO/OSI or IEEE 802. In this context, efforts that include too much of the database functionality inside the operating system kernel or those that modify tightly-closed operating systems are likely to prove unsuccessful. In our view, the operating system should only implement the essential OS services and those DBMS functions that it can *efficiently* implement and then *should get out of the way*. The model that best fits this requirement seems to be the client-server architecture with a small kernel that provides the database functionality that can efficiently be provided and does not hinder the DBMS in efficiently implementing other services at the user level (e.g., Mach [RASH89], Amoeba [TANE90]). Object-orientation may also have a lot to offer as a system structuring approach to facilitate this integration.

Naming is the fundamental mechanism that is available to the operating system for providing transparent access to system resources. Whether or not access to distributed objects should be transparent at the operating system level is a contentious issue. Proponents of transparency argue that it simplifies the user access to distributed objects, that it enables free movement of objects around the system, and that it facilitates load balancing. Opponents, on the other hand, point to the overhead of providing transparent access and claim that it is not reasonable to expect applications that do not require transparency to suffer the performance implications. They also indicate that it is not necessary to have transparency in order to have a coherent view of the system. This discussion is likely to continue for some time. From the perspective of a distributed DBMS, however, transparency is important. As we indicated in Section 3.1, many of the existing distributed DBMSs attempt to establish their own transparent naming schemes without significant success. More work is necessary in investigating the naming issue and the relationship between distributed directories and OS name servers. A worthwhile naming construct that deserves some attention in this context is the capability concept which was used in older systems such as Hydra [WULF81] and is being used in more modern OSs such as Amoeba [TANE90].

Storage and management of *persistent data* which survive past the execution of the program that manipulates them is the primary function of database management systems. Operating systems have traditionally dealt with persistent data by means of file systems. If a successful cooperation paradigm can be found, it may be possible to use the DBMS as the OS file system. At a more general level, the cooperation between programming languages, DBMS and OS to manage persistent data requires further research. There are (at least) three dimensions in addressing the persistent data management issues: problems that DBMSs have with traditional file systems, issues in supporting persistence within the context of database programming languages and how such languages may impact operating system functionality, and specific considerations of distributed file systems. From the unique perspective of this paper, the third dimension is the more important one.

The distributed file systems operate in one of the following three modes [GIFF88]: (1) *remote-*

*disk systems*, which simply enable the sharing of unstructured remote disk blocks among a number of clients; (2) *block-level-access systems*, which enable the accessing of portions of remote files (called *file blocks*); and (3) *file-level-access systems*, which enable sharing of entire files among clients by transferring them from one machine to another.

These distributed file system organizations are not suitable for distributed DBMS implementation. Remote disk systems are not appropriate because they do not permit any level of concurrency in updating data, which is a significant feature of any state-of-the-art distributed DBMS. Each remote disk block has to be marked as either read-only or mutable, and only the read-only blocks can be concurrently accessed. Similarly, file-level-access systems provide concurrent access only at the file level. In a relational distributed DBMS where each file stores a relation, this means locking out an entire relation which would have significant performance penalties. Even though block-level-access systems permit concurrent access at the block level, this eliminates the possibility of implementing predicate locks which only lock-out the accessed records in a block rather than the entire block. Commercial DBMSs do not implement predicate locking, but provide page-level locking instead. This is due to the performance overhead of record-level locking in relational systems. However, the ability to lock at a level of granularity smaller than a page becomes important in object-oriented systems where each page stores multiple objects. Additionally, these remote file system implementations rely on transmitting either fixed size blocks or entire files between machines. However, the results of distributed DBMS queries typically vary in size and the query processor attempts to minimize the amount of data transmission. It is clear that there is a need for some mode of communication between the query processor and the distributed file system as well as new paradigms of implementing remote file access.

Two communication paradigms that have been widely studied in distributed operating systems are message passing[4] and RPC. The relative merits of each approach have long been debated, but the simple semantics of RPC (blocking, one time execution) have been appealing to distributed system designers. As discussed in Section 3.1, an RPC-based access to distributed data at the user level is sometimes proposed in place of providing fully transparent access [GRAY89]. However, implementation of an RPC mechanism for a heterogeneous computing environment is not an easy matter. The issue is that the RPC systems of different vendors do not interoperate. For example, even though both Sun and Apollo workstations are Unix-based[5] and even though both implement RPC mechanisms based on IP-class network protocols, they cannot interoperate. Among other

[4]Note that we are referring to logical message passing, not to physical. Remote procedure calls have to be transmitted between sites as physical messages as well.
[5]Unix is a trademark of AT&T Bell Laboratories.

reasons, one may cite their incompatible communication paradigms: one uses virtual circuits (i.e., supports session-based communication) while the other is datagram-based. It can be argued that this type of interoperability is at the level of the distributed operating system and does not necessarily concern the distributed DBMS. This is true to a certain extent, and the issue has been studied at that level [STOY90, RENE87]. On the other hand, the incompleteness of RPC, even with its simple and restricted communication semantics, needs to be emphasized. It may be necessary to look at communication at higher levels of abstraction in order to overcome heterogeneity or at lower levels of abstraction (i.e., message passing) to achieve more parallelism. This tradeoff needs to be further studied.

In current DBMSs, the transaction manager is implemented as part of the DBMS. Whether transactions should and can be implemented as part of standard operating system services has long been discussed. It is fair to state that there are strong arguments on both sides, but a clear resolution of the issue requires more research as well as some more experience with the various general purpose (i.e., non-DBMS) transaction management services. Some example systems include Camelot [EPPI91] which adds transaction management services to Mach [RASH89], QuickSilver [HASK88] which includes transaction management as standard operating system service and Argus [LISK88] where the transaction concept is supported as a language primitive.

A detailed discussion of the pros and cons of providing an operating system transaction management service is beyond the scope of this paper (see [WEIK86] for a discussion). However, any OS transaction mechanism needs to be sufficiently powerful to support abstract operations on abstract objects [WEIH88]. In this context, the multilevel transaction model discussed earlier may be appropriate. The operating system can efficiently implement the lowest level of abstraction and let the user level applications extend this abstraction to a model that is suitable for their domain. For example, page level locking can be one abstraction level while predicate locking may be another level. The operating system then has to provide the mechanisms that enable the definition of successive levels of abstractions. An added advantage of such an approach may be that nested transactions, which are necessary for complex application domains, can be more easily reconciled with the multilevel transaction model.

## 5. CHANGING TECHNOLOGY AND NEW ISSUES

Distribution is commonly identified as one of the major features of next generation database systems which will be ushered in by the penetration of database technology into new application areas with different requirements than traditional business data processing and the technological developments in computer architecture and networking. The question, from the perspective of this paper, is what will be the nature of distribution in these systems. Answering this question requires the definition of the features of next generation DBMSs, which is a controversial issue. There are a number of documents that attempt to provide such definitions [SILB90, ATKI89, STON90a, STON90b] from which it is possible to identify certain certain common features:

1. *Data model.* One of the common features is that the data model to be supported will need to be more powerful than the relational model, yet without compromising its advantages (data independence and high-level query languages). When applied to more complex application domains such as CAD/CAM, software design, office information systems, and expert systems the relational data model exhibits limitations in terms of complex object support, type system and rule management. To address these issues, two important technologies, knowledge bases and object-oriented databases, are currently being investigated.

2. *Parallellism.* A major issue that faces designers of next generation DBMS is going to be system performance as more functionality is added. Exploiting the parallelism available in multiprocessor computers is one promising approach to provide high performance. Techniques designed for distributed databases can be useful but need to be significantly extended to implement parallel database systems.

3. *Interoperability.* A consequence of applying database technology to various application domains may well be a proliferation of different, yet complementary, DBMSs and more generally information systems. Thus, an important direction of research is interoperability of such systems within a computer network. Multidatabase system technology requires significant extensions towards this goal.

In the following sections, we discuss each of these topics and highlight the research issues that have started receiving attention in the database community.

## 5.1. Distributed Object-Oriented Database Systems

Object-oriented databases (OODBs) [ATKI89, KIM89, ZDON90] combine object-oriented programming (OOP) and database technologies in order to provide higher modeling power and flexibility to data intensive application programmers. There is controversy as to the nature of an object-oriented data model, i.e., whether it should be an extension of the relational model with nested relations (e.g., DASDBS [SCHE90]) or abstract data types (e.g., POSTGRES [STON90c]), or a new model borrowing from OOP concepts (e.g., $O_2$ [LECL88], ORION [KIM90], GEMSTONE [MAIE87]). In any case, in addition to traditional database functions, the primary functions to be supported are abstract data types, type inheritance, and complex objects.

Over the last five years, OODBs have been the subject of intensive research and experimentation which led to an impressive number of prototypes and commercial products. However, the theory and practice of developing distributed object-oriented DBMSs (OODBMSs) have yet to be fully developed. This effort almost parallels that of relational systems. Even though some of the solutions developed for relational systems are applicable, the high degree of generality introduced by the object-oriented data models creates significant difficulties. In this section we review the more important issues related to the system architecture, query models and processing, transaction processing and distributed object management.

**Distributed OODBMS Architecture**

Dealing with distributed environments will make the problems even more difficult. Additionally, the issues related to data dictionary management and distributed object management have to be dealt with. However, distribution is an essential requirement, since applications which require OODB technology typically arise in networked workstation environments. The early commercial OODBMSs (e.g., GEMSTONE) use a client/server architecture where multiple workstations can access the database centralized on a server. However, as a result of hardware advances in microprocessor technology, the processing power of such distributed systems gets concentrated in the client workstations rather than the server [DEWI90b]. A similar observation can be made for the capacity and speed of the local disks attached to workstations compared to the larger server's disks. Consequently, distributing an OODB within a network of workstations (and servers) becomes very attractive. In fact, some OODBMSs already support some form of data distribution transparency (e.g., ONTOS and Distributed ORION).

The design of distributed OODBMS should benefit from the experience gained from distributed databases. In particular, if the system is an extension of a relational DBMS, then most distributed database solutions can apply. However, in the case of a more general object-oriented data model, the management of distributed objects is complicated by the transparency and efficiency requirements. Many distributed object-oriented systems simplify the problem by making the user aware of object location and performing remote accesses to non-local objects. This approach is not appropriate for distributed OODBMSs because location transparency and distributed program execution, two essential functions for high performance and availability, would be precluded.

**Queries and Query Processing**

In order not to compromise the obvious advantages of relational systems, an OODBMS ought to provide a high-level query language for object manipulation. While there has been some proposals for calculus and algebras for OODBs, query optimization remains an open issue. OODB queries are more complicated and can include path traversals and ADT operations as part of their predicates. The initial work on query processing (e.g., [OSBO88, KORT88, GRAE88, SHAW89, STRA90]) does not consider object distribution. The efficient processing of distributed OODB queries can borrow from distributed relational query processing to exploit the fragmentation of collection objects. However, achieving correct program transformations is more involved due to the higher expressive power of object-oriented query languages, especially in the presence of objects that are referentially shared [KHOS87].

### Transaction Management

Difficulties are introduced in transaction management for three reasons. First, objects can be complex thereby making variable-granularity locking essential. Second, support for dynamic schema evolution requires efficient solutions for updating schema data. Third, to address the different requirements of the targeted application domains, several concurrency control algorithms need be supported (e.g., pessimistic and optimistic concurrency control as in GEMSTONE). Furthermore, engineering applications typically require specific support for cooperative transactions or long-duration nested transactions. In Section 4.4 we indicated the possible transaction models. In object-oriented systems, full generality is typically required such that complex transactions operate on complex object structures. Furthermore, the object model may treat transactions as first class objects, both adding complexity and more opportunities to support multiple transaction types in one system. This is an area which has started to attract some attention, but is still in its infancy.

Replication of objects, replica control protocols and their relationship to concurrency control algorithms in object-oriented systems opens some interesting issues. The information hiding implicit in typed data offers an opportunity for the storage system (including the replication algorithms) of the systematic exploitation of the type semantics. An example of a type-specific approach is that of *event-based data replication* [HERL86, HEDD88], in which the unit of replication is a single operation execution, and not the entire value of the object. Thus, a replica contains a (possibly partial) history or log of operation execution and checkpoint events, rather than a value or a set of versions. By contrast, traditional (value-based) data replication methods maintain copies of object values, with an associated log that is used exclusively for local recovery from site failures. Event-based data replication promises more efficient utilization of disks because log writes eliminate seek times, higher concurrency by permitting type-specific concurrency control algorithms to operate directly on the log representation, and higher availability and faster response time for common update operations that are semantically blind-writes (log append). In addition, this approach forms a basis for the seamless integration of type-specific and type-independent algorithms for concurrency control, recovery, and replication, in the same system. This is because the event log degenerates into a value if the checkpoints are constrained to be always current.

**Distributed Object Management**

Efficient management of objects with complex connections is difficult. When objects can be hierarchical and contain possibly shared subobjects, object clustering and indexing is a major issue [KEMP90]. With object sharing, garbage collection of objects which become orphans after updates is problematic. Furthermore, large-size objects such as graphics and images need special attention, especially in distributed systems where objects are moved between the client and server machines. It is important to maintain physical contiguity of objects to reduce the data movement overhead.

Distributed object management should rely on a distributed storage model which can capture the clustering and fragmentation of complex objects [KHOS89, GRUB91]. Solutions developed for relational systems can be applied to collections of objects. However, the main problems remain the support of global object identity and object sharing. Global object identity is expensive because of the current lack of global virtual address spaces. Therefore, the distinction between objects (with OIDs) and values is important since values can be manipulated as in relational systems. Managing distributed shared objects is difficult since inconsistencies can occur when a shared object is moved and updated at another site. Possible solutions include the use of an indirection table or the avoidance of problems at compile time. With the first kind of solutions, distributed garbage collection is an open problem [LISK86, SHAP90].

The development of distributed object-oriented DBMSs bring to the forefront the issues related to proper OS support. The issues are more interesting in this case since the development of object-oriented distributed operating systems has also been studied independently. Object-oriented technology can serve as the common platform to eliminate the "impedance mismatch" between the programming languages, database management systems, and operating systems. The integration of the first two have been addressed by researchers and we have indicated some of this work above. However, the integration of object-oriented DBMSs and object-oriented operating systems have not yet been studied and remains an interesting research issue. One problem in using object-orientation to bring these systems together is their differing understandings of what an object is and their quite different type systems. Nevertheless, if the next generation object-oriented DBMSs are to have an easier cooperation with the operating systems than today's relational DBMSs do, these issues need to be addressed.

## 5.2. Distributed Knowledge Base Systems

An important feature that is likely to be present in next generation DBMSs is some facility for reasoning. This is a feature that should enable us to move from data management to more general knowledge management by abstracting the reasoning mechanism from the application programs and encapsulating it within the DBMS. This does not mean that the reasoning capability needs to be centralized, but only that the responsibility for its application is taken out of the application program. The reasoning engine can be distributed as well as the rules according to which it is applied. In object-oriented knowledge base implementations, for example, reasoning rules can be encapsulated in each object.

Capturing knowledge in the form of rules has been extensively studied in a particular form of knowledge base system called a *deductive database*. Deductive database systems [GALL84] manage and process rules against large amounts of data within the DBMS rather than with a separate subsystem. Rules can be declarative (assertions) or imperative (triggers). By isolating the application knowledge and behavior within rules, knowledge base management systems (KBMSs) provide control over knowledge which can be better shared among users. Furthermore, the high expressive power of rule programs aids application development. These advantages imply increased programmer productivity and application performance. Based on first order logic, deductive database technology subsumes relational database technology. We can isolate two alternative approaches to KBMS design. The first one extends a relational DBMS with a more powerful rule language (e.g., RDL [KIER90] or ESQL [GARD90]), while the second approach extends first order logic into a declarative programming language such as Datalog (e.g., LDL [CHIM90]). The two approaches raise similar issues, some of which have been partially addressed by the artificial intelligence community, although with strong assumptions such as a small database and the absence of data sharing.

Rule management, as investigated in deductive databases, is essential since it provides a uniform paradigm to deal with semantic integrity control, views, protection, deduction and triggers. Much of the work in deductive databases has concentrated on the semantics of rule programs and on processing deductive queries – in particular, in the presence of recursive and negated predicates [BANC86]. However, there are a number of open issues related to the enforcement of the semantic consistency of the knowledge base, optimization of rule programs involving large amounts of data and rules, integration of rule-based languages with other tools (e.g., application generators), and providing appropriate debugging tools. Furthermore, much work is needed to combine rule support with object-oriented capabilities such as class mechanisms and complex objects.

For reasons similar to those for OODB applications, knowledge base applications are likely to arise in networked workstation environments. These applications can also arise in parallel computing environments when the database is managed by a multiprocessor database server (see next section). In any case, there are a number of similar issues which can get simplified by relying on distributed relational database technology. Unlike most OODB approaches which try to extend an OOPL system, this is a strong advantage for implementing knowledge bases in distributed environments. Therefore, the new issues have more to do with distributed knowledge management and processing and debugging of distributed knowledge base queries than with distributed data management.

Distributed knowledge management includes the management of various kinds of rules such as assertions, deductive rules and triggers. The question is how and where they should be stored in order to minimize a cost function. When dealing with fragmented relations, fragment definition and rule definition should be investigated jointly so that rules can efficiently apply to fragments. The preventive approach of compiling assertions, such as semantic integrity constraints, into efficient pretests [SIMO86] can possibly be extended to address more general rules. Another possibility is to implement rule firing based on updates (e.g., triggers) as an extension of the transaction management mechanism [STON90c, DAYA88, MCCA89]. In any case, distributed database design must be revisited to take into account rules when defining fragmentation.

Distributed knowledge base query processing and debugging is made difficult by the presence of additional capabilities, such as recursive rules, and the larger range of distributed execution strategies for such capabilities. Most of the work in this area has focused on extending distributed query processing to support the transitive closure of fragmented relations in parallel [VALD88]. More general Datalog programs are considered for distributed execution in WOLF90]. However, much more work is needed to provide a general framework to process distributed knowledge base queries. As in the centralized case, debugging of distributed knowledge base queries is more difficult due to the complex transformations that can be applied to input queries.

## 5.3. Parallel Database Systems

Parallel database systems intend to exploit the recent multiprocessor computer architectures in order to build high-performance and fault-tolerant database servers [VALD91]. This can be achieved by extending distributed database technology, for example, by fragmenting the database across multiple nodes so that more inter- and intra-query parallelism can be obtained. For obvious reasons such as set-oriented processing and application portability, most of the work in this area has focused on supporting SQL. Bubba [BORA90] is a notable exception, because it supports a more powerful database language. There are already some relational database products that implement this approach, e.g., Teradata's DBC [NECH85] and Tandem's NonStop SQL [TAND87], and the number of such products will increase as the market for general-purpose parallel computers expands. We have started to see implementations of existing relational DBMSs on parallel computers (e.g., INGRES on Encore and ORACLE on Sequent).

The design of parallel database systems faces a number of challenging issues with respect to operating system support, data placement, parallel algorithms, and parallelizing compilation. These issues are common to both kinds of multiprocessor architectures. Furthermore, if the supported language has object-oriented or deductive capabilities, we obviously have the issues previously discussed (e.g., placement of complex objects).

In the context of dedicated database servers, specific operating system support for parallel data management can be very cost-effective. Two approaches can be applied. The first one creates a brand new dedicated operating system from almost scratch, e.g., the Bubba operating system, which implements a single-level store where all data are uniformly represented in a virtual address space [COPE90]. The second one tries to capitalize on modern, open distributed operating systems, e.g., Mach, and extends it in a way that can provide single-level store efficiency with special attention to dataflow support.

Data placement in a parallel database server exhibits similarities with horizontal fragmentation. An obvious similarity is that parallelism can be dictated by fragmentation. It is much more involved in the case of a distributed-memory architecture because load balancing is directly implied by the fragmentation. The solution proposed in [COPE88] insists that data placement (fragmentation and allocation of fragments) is initially based on the size and access frequency of the relations, and can be revised by periodic reorganizations. The main issue here is to decide when and how to perform such reorganizations. Another issue is to avoid to recompile all the queries because of reorganization.

Fragmented data placement is the basis for parallel data processing. The first issue is therefore to design parallel algorithms which can exploit such data placement. The problem is simple for select operations and more difficult for joins [BITT83, VALD84]. More work is needed to study the effect of indexes or dynamic reorganization on parallel algorithms, and their application to other expensive or complex operations, e.g., transitive closure. Furthermore, some capabilities of the interconnect network such broadcasting can be exploited here.

Parallelizing compilation transforms an input query into an efficient low-level equivalent program which can be directly executed by the parallel system. An open issue here is to define a good parallel language in which optimization decisions regarding the parallel algorithms and the dataflow control can be expressed. Parallel FAD [HART88] and Parallel LERA [CHAC91] are first attempts towards this goal. The main issue in language design is to provide a good trade-off between generality and abstraction. Generality is essential to be able to express all kinds of parallel execution strategies supported by the system; abstraction is important to avoid dealing with low-level details of optimization.

If parallel data servers become prevalent, it is not difficult to see an environment where multiple of them are placed on a backbone network. This gives rise to distributed systems consisting of clusters of processors [GRAY89]. An interesting concern in such an environment is internetworking. Specifically, the execution of database commands which span multiple, and possibly heterogeneous, clusters creates at least the problems that we discuss under distributed multidatabase systems in the next section. However, there are the additional problems that the queries have to be optimized not only for execution in parallel on a cluster of servers, but also for execution across a network.

## 5.4. Distributed Multidatabase Systems

As we indicated in Section 2, multidatabase system organization is an alternative to logically integrated distributed databases. The fundamental difference between the two approaches is the level of autonomy afforded to the component data managers at each site. While integrated DBMSs have components which are designed to work together, multidatabase management systems (multi-DBMS) consist of components which may not have any notion of cooperation. Specifically, the components are independent DBMSs, which means, for example, that while they may have facilities to execute transactions, they have no notion of executing distributed transactions that span multiple components (i.e., they do not have global concurrency control mechanisms or distributed commit protocol implementation).

As an alternative architectural model, distributed multidatabase systems share the problems of distributed DBMSs and introduce additional ones of their own. As such, they probably deserve more space and attention than we are able to devote in this paper. In this section, we briefly highlight the open problems that relate to global schema definition and management, query processing and transaction management.

### Global Schema Definition and Management Issues

The arguments against full transparency (see Section 3.1) gain much more credibility in multidatabase systems. The additional autonomy of individual components and their potential heterogeneity make it more difficult (some claim impossible) to support full transparency. A major difficulty relates to the definition of the global conceptual schema (GCS) as a specification of the content and structure of the global database. The definition and role of the GCS is well understood in the case of integrated distributed database systems: it defines a logical single image over physically distributed data. The same understanding does not exist in the case of multidatabase systems, however. First, the global conceptual schema may or may not define the entire database since each system may autonomously decide which parts of its local database it wants to make globally accessible. Second, the definition of a global conceptual schema from a large number of frequently changing local schemas is considered to be very difficult [SCHE90b]. Third, the maintenance of consistency of the global conceptual schema is a significant problem in multidatabase systems. It is quite possible for local schemas to be modified independently such that the global schema definition is affected. In this case, the maintenance of the consistency of the GCS requires that it also be modified according to the changes to the local schema. Setting up the mechanisms for mapping these changes is not easy. One way the issue can be attacked is to treat the global conceptual schema not as a concrete definition, but a generalization view defined over local conceptual schemas. Studies along this line have been conducted before [MOTR81, DAYA84] and their practical implications with respect to transparency need to be considered. Finally, the definition and enforcement of global integrity constraints need to be addressed. Most current research ignores data dependencies across autonomous databases.

If the global conceptual schema can be defined as a single view over the local database schemas, why can't multiple views for different applications be defined over the same local schemas? This observation has led some researchers to suggest that a multi-DBMS be defined as a system that manages “several databases without a global schema” [LITW88]. It is argued that the absence of a GCS is a significant advantage of multidatabase systems over distributed database systems.

An example of integrating multiple databases without defining a global conceptual schema is MRDSM which specifies a language (called MDL [LITW87]) with sufficiently powerful constructs to access multiple databases. A significant research problem is the nature of such a language. As indicated in [SCHE90b], the component DBMSs may implement special operators in their query languages which makes it difficult to define a language that is as powerful as the union of the component ones. It is also possible that component DBMSs may provide different interfaces to general purpose programming languages. The solution may be to implement languages which are extensible.

## Query Processing

The autonomy and potential heterogeneity of component systems create problems in query processing and especially in query optimization. The fundamental problem is the difficulty of global optimization when local cost functions are not known and local cost values cannot be communicated to the multi-DBMS. It has been suggested that semantic optimization based only on qualitative information may be the best that one can do [SCHE90b], but semantic query processing is not fully understood either. Potentially, it may be possible to develop hierarchical query optimizers which perform some amount of global query optimization and then let each local system perform further optimization on the localized subquery. This may not provide an "optimal" solution, but may still enable some amount of optimization. The emerging standards, which we will discuss shortly, may also make it easier to share some cost information.

## Transaction Processing

Transaction processing in autonomous multidatabase systems is another important research area. The emerging consensus on the transaction execution model seems to be the following. Each component DBMS has its own transaction processing services (i.e., transaction manager, scheduler, recovery manager) and is capable of accepting *local transactions* and running them to completion. In addition, the multi-DBMS layer has its own transaction processing components (global transaction manager, global scheduler) in charge of accepting *global transactions* accessing multiple databases and coordinating their execution.

Autonomy requires that the global transaction management functions be performed independently of the local transaction execution functions. In other words, the individual DBMSs (more specifically, the transaction processing components) or their local applications are not modified to accommodate global transactions. Heterogeneity has the additional implication that the transaction processing components may employ different concurrency control, commit, and recovery protocols.

In this model of execution, the global transactions are broken down into a set of *subtransactions* each of which is submitted to a component DBMS for execution. Local DBMSs execute these subtransactions intermixed with the local transactions that they receive directly. The existence of local transactions that are unknown to the global scheduler complicates things and may create *indirect conflicts* among global transactions. Furthermore, the subtransactions that are submitted to a component DBMS on behalf of a global transaction are treated as local transactions by the component DBMSs. Therefore, there is the additional problem of ensuring that the ordering imposed upon the global transactions by the global scheduler is maintained for all of their subtransactions at each component DBMS by all the local schedulers [BREI88]. If serializability is used as the correctness criterion for concurrent execution of transactions, the problems is ensuring that the serialization order of global transactions at each site are the same.

A number of different solutions have been proposed to deal with concurrent multidatabase transaction processing. Some of these use global serializability of transactions as their correctness criteria (e.g., [GEOR91, BREI88]) while others relax serializability (e.g., [DU89, BARK90]). Most of this work should be treated as being preliminary initial attempts at understanding and formalizing the issues. There are many issues that remain to be investigated. One area of investigation has to deal with revisions in the transaction model and the correctness criteria. There are initial attempts to recast the transaction model assumptions [GARC90b, ELMA90] and this work needs to continue. Nested transaction models look particularly promising for multidatabase systems and its semantics based on knowledge about the transaction's behavior (similar to [FARR89]) needs to be formalized. In this context, it is necessary to go back and reconsider the meaning of consistency in multidatabase systems. A good starting point is the four degrees of consistency defined by Gray [GRAY79].

Another difficult issue that requires further investigation is the reliability and recovery aspects of multidatabase systems. The above described transaction execution model is quite similar to the execution of distributed transactions. However, if the component DBMSs are fully autonomous, then they do not incorporate 2PC procedures which makes it difficult to enforce distributed transaction atomicity. Even though the topic has been discussed in some recent works [BARK91, GEOR90, WOLS90], these approaches are initial engineering solutions. The development of reliability and recovery protocols for multidatabase systems and their integration with concurrency control mechanisms still needs to be worked out.

### Autonomy and Standards

Probably one the fundamental impediments to further development of multidatabase systems is the lack of understanding of the nature of autonomy, which plays a major role in making life miserable. It is probably the case that what we call “autonomy” is itself composed of a number of factors. An initial characterization that has been made identifies three different forms of autonomy: design autonomy, communication autonomy, and execution autonomy. Other characterizations have also been proposed [GARC88, SHET90]. These characterizations need to be worked out more precisely and exhaustively. Furthermore, most researchers treat autonomy as if it were a “all-or-nothing” feature. Even the taxonomy that we considered in Section 2 indicates only three points along this dimension. However, the spectrum between “no autonomy” and “full autonomy” probably contains many distinct points. It is essential, in our opinion, to (1) precisely define what is meant by “no autonomy” and “full autonomy”, (2) precisely delineate and define the many different levels of autonomy, and (3) identify, as we indicated above, the appropriate degree of database consistency that is possible for each of these levels. At that point, it would be more appropriate to discuss the different transaction models and execution semantics that are appropriate at each of these levels of autonomy. In addition, this process should enable the identification of a layered structure, similar to the ISO Open System Interconnection model, for the interoperability of autonomous and possibly heterogeneous database systems. Such a development would then make it possible to specify interfacing standards at different levels. Some standardization work is already under way within the context of the Remote Data Access (RDA) standard, and this line of work will make the development of practical solutions to the interoperability problem possible.

**Object-Orientation and Multidatabase Systems**

Another important ingredient is the development of the technology that can inherently deal with autonomous software components. A prime candidate technology is object-orientation. We discussed its role in the development of database management systems in Section 5.1. Here we only comment on its role in addressing the interoperability issues.

Object-oriented systems typically treat the entities in their domain as instances of some abstract type. In the case of database systems, these entities are usually data objects. However, it is quite common for object-oriented models to treat every entity uniformly as objects. Thus, for example, programs, user queries, and so on are considered objects in certain object models. The technology is primarily concerned with providing the necessary tools and methodology for the consistent management of these entities while there are user accesses (both to retrieve and to update) to them. This is to be done without allowing the user to “see” the physical structure of each object. In other words, user access to objects is only by means of well-defined interfaces which hide their internal structure. In the case of applying object-orientation to the interoperability problem, one considers each autonomous DBMS to be an object. The relationship is then established between manipulating objects without “looking inside them” and being able to perform operations on autonomous DBMSs without “requiring changes in them.” This is a very simple characterization of the problem. There has been some initial discussion of the potential benefits that may be gained from the application of the object-oriented technology to the interoperability problem [MANO90, SCHE90b, HEIL90] and this is a major direction that needs to be further investigated.

## 6. CONCLUSIONS

In this paper we discuss the state-of-the-art in distributed database research and products. Specifically, we address the following questions:

1. What were the initial goals and promises of the distributed database technology? How do the current commercial products measure up to these promises? In retrospect, were these goals achievable?

2. Have the important technical problems already been solved?

3. What are the technological changes that underlie distributed data managers and how will they impact next generation systems?

The initial promises of distributed database systems, namely transparent management of distributed and replicated data, improved system reliability via distributed transactions, improved system performance by means of inter- and intra-query parallelism, and easier and more economical system expansion, are met to varying degrees by existing commercial products. Full realization of these promises is not only dependent upon the commercial state-of-the-art catching up with the research results, but is also dependent upon the solution of a number of problems. The issues that have been studied, but still require more work are the following:

1. performance models, methodologies and benchmarks to better understand the sensitivity of the proposed algorithms and mechanisms to the underlying technology;

2. distributed query processing to handle queries which are more complex than select-project-join, ability to process multiple queries at once to save on common work, and optimization cost models to be able to determine when such multiple query processing is beneficial;

3. advanced transaction models that differ from those that are defined for business data processing and that better reflect the mode of processing that is common in most distributed applications (i.e., cooperative sharing vs competition, interaction with the user, long duration) for which the distributed database technology is going to provide support;

4. analysis of replication and its impact on distributed database system architecture and algorithms and the development of efficient replica control protocols that improve system availability;

5. implementation strategies for distributed DBMSs that emphasize a better interface and co-

operation with distributed operating systems; and

6. theoretically complete and correct and practical design methodologies for distributed databases.

In addition to these existing problems, we identify the changing nature of the technology on which distributed DBMSs are implemented, and discuss the issues that need to be investigated to facilitate the development of next generation distributed DBMSs. Basically, we expect the following changes:

1. Advances in the development of cost-effective multiprocessors will make parallel database servers feasible. This will effect DDBSs in two ways. First, distributed DBMSs will be implemented on these parallel database servers, requiring the revision of most of the existing algorithms and protocols to operate on the parallel machines. Second, the parallel database servers will be connected as servers to networks, requiring the development of distributed DBMSs that will have to deal with a hierarchy of data managers.

2. As distributed database technology infiltrates non-business data processing type application domains (e.g., engineering databases, office information systems, software development environments), the capabilities required of these systems will change. This will necessitate a shift in emphasis from relational systems to data models which are more powerful. Current research along these lines concentrates on object-orientation and knowledge base systems.

3. The above mentioned diversity of distributed database application domains is probably not possible through a tightly-integrated system design. Consequently, distributed database technology will need to effectively deal with environments which consist of a federation of autonomous systems. The requirement for interoperability between autonomous and possibly heterogeneous systems has prompted research in multidatabase systems.

As this paper clearly demonstrates, there are many important technical problems that await solution, as well as new ones that arise as a result of the technological changes that underlie distributed data managers. These problems should keep researchers as well as distributed DBMS implementers quite busy for some time to come.

## ACKNOWLEDGMENTS

We would like to extend our thanks to Abdelsalam Heddaya of Boston University who not only reviewed the entire paper and provided many comments, but also helped us with Section 4.3.

In fact, the replication part of that section is written by him. Alex Biliris of Boston University, Michael Brodie, Alex Buchmann, Dimitrios Georgakopoulos, and Frank Manola of GTE Laboratories have also read the entire manuscript and provided many suggestions regarding the content and the presentation that improved the paper significantly.

The research of the first author has been partially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada under operating grant OGP-0951.

## REFERENCES

[ALSB76] P. ALSBERG, G. BELFORD, J. DAY, AND E. GRAPA. *Mutli-Copy Resiliency Techniques*, CAC Document 202, Center for Advanced Computation, University of Illinois, Urbana, Illinois, May 1976.

[APER81] P.M.G. APERS. “Redundant Allocation of Relations in a Communication Network,” In *Proc. 5th Berkeley Workshop on Distributed Data Management and Computer Networks*, Berkeley, Calif., 1981, pp. 245–258.

[ATKI89] M. ATKINSON, F. BANCILHON, D.J. DEWITT, K. DITTRICH, D. MAIER, AND S. ZDONIK. “The Object-Oriented Database System Manifesto,” In *Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases,* Kyoto, Japan, 1989, pp. 40–57.

[AVIZ85] A. AVIZIENIS. “The N-Version Approach to Fault-Tolerant Software,” *IEEE Trans. on Software Eng*. (December 1985), SE-11(12): 1491–1501.

[BADR88] B.R. BADRINATH AND K. RAMAMRITHAM. “Synchronizing Transactions on Objects,” *IEEE Trans. Comp*. (May 1988) 37(5): 541–547.

[BANC86] F. BANCILHON AND R. RAMAKRISHNAN. “An Amateur's Introduction to Recursive Query Processing,” In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Washington D.C., May 1986, pp. 16–52.

[BARK91] K. BARKER AND M.T. ÖZSU. “Reliable Transaction Execution in Multidatabase Systems,” In *Proc. 1st Int. Workshop on Interoperability in Multidatabase Systems*, Kyoto, Japan, 1991, pp. 344–347.

[BARK90] K. BARKER. *Transaction Management on Multidatabase Systems*. Ph.D. Thesis, Edmonton, Alberta, Canada: Department of Computing Science, University of Alberta, 1990 (available as Technical Report TR90-23).

[BEER88] C. BEERI, H.-J. SCHEK, AND G. WEIKUM. “Multilevel Transaction Management, Theoretical Art or Practical Need?” In *Advances in Database Technology – EDBT '88*, J.W. Schmidt, S. Ceri, and M. Missikoff (eds.), New York: Springer-Verlag, 1988, pp. 134–154.

[BEER87] C. BEERI, P.A. BERNSTEIN, AND N. GOODMAN. “A Model for Concurrency in Nested Transaction Systems,” *J. ACM* (April 1989), 36(2): 230–269

[BERN90] P.A. BERNSTEIN. “Transaction Processing Monitors,” *Comm. ACM* (November

1990), 3(11): 75–86.

[BERN84] P.A. BERNSTEIN AND N. GOODMAN. "An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases," *ACM Trans. on Database Syst.* (December 1984), 9(4):596–615.

[BERN81] P.A. BERNSTEIN, N. GOODMAN, E. WONG, C.L. REEVE, AND J.B. ROTHNIE, JR. "Query Processing in a System for Distributed Databases (SDD-1)," *ACM Trans. Database Syst.* (December 1981), 6(4): 602–625.

[BIRM87] K. BIRMAN AND T.A. JOSEPH. "Exploiting Virtual Synchrony in Distributed Systems," In *Proc. 11th ACM Symp. on Operating System Principles*, Austin, TX, November 1987.

[BITT83] D. BITTON, H. BORAL, D.J. DEWITT, AND W.K. WILKINSON. "Parallel Algorithms for the Execution of Relational Database Operations," *ACM Trans. Database Syst.* (September 1983), 8(3): 324–353.

[BORA90] H. BORAL, W. ALEXANDER, L. CLAY, G. COPELAND, S. DANFORTH, M. FRANKLIN, B. HART, M. SMITH, AND P. VALDURIEZ. "Prototyping Bubba, a Highly Parallel Database System," *IEEE Trans. Knowledge and Data Eng.* (March 1990), 2(1): 4–24.

[BORA83] H. BORAL AND D.J. DEWITT. "Database Machines: An Idea Whose Time has Passed? A Critique of the Future of Database Machines," In *Proc. 3rd Int. Workshop on Database Machines*, Munich: Springer-Verlag, 1983, pp. 166–187.

[BREI88] Y. BREITBART AND A. SILBERSCHATZ. Multidatabase Update Issues, In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Chicago, June 1988, pp. 135–142.

[BUCH91] A. BUCHMANN, M.T. ÖZSU, M. HORNICK, D. GEORGAKOPOULOS AND F.A. MANOLA. "A Transaction Model for Active, Distributed Object Systems," In *Advanced Transaction Models for New Applications*, A. Elmagarmid (ed.), San Mateo, Calif.: Morgan Kaufmann, 1991 (in preparation).

[CERI87] S. CERI, B. PERNICI, AND G. WIEDERHOLD. "Distributed Database Design Methodologies," *Proc. IEEE* (May 1987), 75(5): 533–546.

[CHAC91] C. CHACHATY, P. BORLA-SALAMET, AND B. BERGSTEN. "Capturing Parallel Data Processing Strategies within a Compiled Language, " In [VALD91].

[CHIM90] D. CHIMENTI, R. GAMBOA, R. KRISHNAMURTHY, S. NAQVI, S. TSUR, AND C. ZANIOLO. "The LDL System Prototype," *IEEE Trans. Knowledge and Data Eng.* (March 1990), 2(1): 76–90.

[CHRI87] P. CHRISTMANN, TH. HÄRDER, K. MEYER–WEGENER, AND A SIKELER. "Which Kinds of OS Mechanisms Should Be Provided for Database Management," In *Experiences with Distributed Systems*, J. Nehmer (ed.), New York: Springer-Verlag, 1987, pp. 213–251.

[COOP85] E. COOPER. "Replicated Distributed Programs," In *Proc. 10th ACM Symp. on*

*Operating System Principles*, Orcas Island, WA, December 1985, pp. 63–78.

[COPE90] G. COPELAND, M. FRANKLIN, AND G. WEIKUM. "Uniform Object Management," In *Advances in Database Technology – EDBT '90*, Venice, Italy, April 1990, pp. 253–268.

[COPE88] G. COPELAND, W. ALEXANDER, E. BOUGHERTY, AND T. KELLER. "Data Placement in Bubba," In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Chicago, May 1988, pp. 99–108.

[DAYA88] U. DAYAL, A. BUCHMANN, AND D. MCCARTHY. "Rules are Objects Too: A Knowledge Model for an Active Object-Oriented Database System," In *Proc. 2nd Int. Workshop on Object Oriented Database Systems*, Bad Muenster, Germany, 1988, pp. 129–143.

[DAYA84] U. DAYAL AND H. Y. HWANG. "View Definition and Generalization for Database System Integration in a Multidatabase Systeem," *IEEE Trans. on Software Eng.* (November 1984), SE-10(6):628–645.

[DEWI90a] D.J. DEWITT AND J. GRAY. "Parallel Database Systems," *ACM SIGMOD Record* (December 1990), 19(4): 104–112.

[DEWI90b] D.J. DEWITT, P. FUTTERSACK, D. MAIER, AND F. VELEZ. "A Study of Three Alternative Workstaion Server Architectures for Object-Oriented Database Systems," In *Proc. 16th Int. Conf. on Very Large Data Bases*, Brisbane, Australia, August 1990, pp.107–121.

[DU89] W. DU AND A. ELMAGARMID. "Quasi-Serializability: A Correctness Criterion for Global Concurrency Control in InterBase," In *Proc. 15th Int. Conf. on Very Large Data Bases*, Amsterdam, August 1989, pp. 347–355.

[ELMA90] A.K. ELMAGARMID, Y. LEU, W. LITWIN, AND M. RUSINKIEWICZ. "A Multidatabase Transaction Model for InterBase," In *Proc. 16th Int. Conf. on Very Large Data Bases*, Brisbane, Spain, August 1990, pp. 507–518.

[EPPI91] J.L. EPPINGER, L.B. MUMMERT AND A. SPECTOR (eds.). *Camelot and Avalon, A Distributed Transaction Facility*, San Mateo, CA: Morgan Kaufmann, 1991.

[FARR89] A.A. FARRAG AND M.T. ÖZSU. "Using Semantic Knowledge of Transactions to Increase Concurrency," *ACM Trans. on Database Syst.* (December 1989), 14(4): 503–525.

[FREY87] C. FREYTAG. "A Rule-based View of Query Optimization," In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, San Francisco, 1987, pp. 173– 180.

[GALL84] H. GALLAIRE, J. MINKER, AND J-M. NICOLAS. "Logic and Databases: A Deductive approach," *ACM Comput. Surv.* (June 1984), 16(2): 153–186.

[GARC90a] H. GARCIA-MOLINA AND B. LINDSAY. "Research Directions for Distributed Databases," *IEEE Q. Bull. Database Eng.* (December 1990), 13(4): 12–17.

[GARC90b] H. GARCIA-MOLINA, D. GAWLICK, J. KLEIN, K. KLEISSNER, AND K. SALEM.

Coordinating Multi-Transaction Activities. Technical Report CS-TR-247-90, Princeton, NJ: Princeton University, February 1990.

[GARC88] H. GARCIA-MOLINA AND B. KOGAN. "Node Autonomy in Distributed Systems," In *Proc. Int. Symp. on Databases in Parallel and Distributed Systems*, Austin, TX, December 1988.

[GARC87] H. GARCIA-MOLINA AND K. SALEM. "Sagas," *Proc. ACM SIGMOD Int. Conf. on Management of Data,* San Fransisco, CA, 1987, pp. 249-259.

[GARD90] G. GARDARIN AND P. VALDURIEZ, "ESQL: An Extended SQL with Object and Deductive Capabilities," In *Proc. Int. Conf. on Database and Expert System Applications*, Vienna, Austria, August 1990.

[GEOR91] D. GEORGAKOPOULOS. "Multidatabase Recoverability and Recovery," In *Proc. 1st Int. Workshop on Interoperability in Multidatabase Systems*, Kyoto, Japan, 1991, pp. 348–355.

[GEOR90] D. GEORGAKOPOULOS. *Transaction Management in Multidatabase Systems*. Ph.D. thesis, Houston, TX: Department of Computer Science, University of Houston, 1990.

[GIFF88] D.K. GIFFORD, R.M. NEEDHAM, AND M.D. SCHROEDER. "The Cedar File System," *Comm. ACM* (March 1988), 31(3): 288-298.

[GIFF79] D.K. GIFFORD. Weighted Voting for Replicated Data. In *Proc. 7th ACM Symp. on Operating System Principles*, Pacific Grove, Calif., December 1979, pp. 150–159.

[GRAE88] G. GRAEFE AND D. MAIER. "Query Optimization in Object-Oriented Database Systems: A Prospectus," In *Advances in Object Oriented Database Systems*, K. R. Dittrich (ed.), New York, NY: Springer-Verlag, 1988, pp. 358–363.

[GRAY89] J. GRAY. *Transparency in its Place – The Case Against Transparent Access to Geographically Distributed Data*, Technical Report TR89.1, Tandem Computers Inc., Cupertino, CA, 1989.

[GRAY85] J. GRAY. *Why Do Computers Stop and What Can Be Done About It*. Technical Report 85-7, Cupertino, Calif.: Tandem Computers, 1985.

[GRAY79] J. GRAY. "Notes on Data Base Operating Systems," In *Operating Systems: An Advanced Course*, R. Bayer, R. M. Graham, and G. Seegmüller (eds.), New York: Springer-Verlag, 1979, pp. 393–481.

[GRIE88] I. GRIEF (ed.). *Computer-Supported Cooperative Work: A Book of Readings*, San Mateo, CA: Morgan Kaufmann, 1988.

[GRUB91] O. GRUBER AND P. VALDURIEZ. "Object Management in Parallel Database Servers" in [VALD91].

[HART88] B. HART, S. DANFORTH, AND P. VALDURIEZ. "Parallelizing FAD: A Database Programming Language," In *Proc. Int. Symp. on Databases in Parallel and Distributed Systems*, Austin, Tex., December 1988, pp. 72–79.

[HASK88] R. HASKIN, Y. MALACHI, W. SAWDON, AND G. CHAN. "Recovery Management in QuickSilver," *ACM Trans. Computer Syst.* (February 1988), 6(1): 82-108.

[HEDD88] A.A. HEDDAYA. *Managing Event-Based Replication for Abstract Data Types in Distributed Systems*. PhD Thesis, Harvard University, Aiken Computation Laboratory, 1988 (available as technical report TR-20-88).

[HEIL90] S. HEILER AND S. ZDONIK. "Object Views: Extending the Vision," In *Proc. Sixth Int. Conf. on Data Eng.,* Los Angeles, CA, 1990, pp. 86–93.

[HERL86] M. HERLIHY. A Quorum-Consensus Replication Method. *ACM Trans. on Comp. Syst.* (February 1986), 4(1): 32–53.

[IOAN90] Y.E. IOANNIDIS AND Y.C. KANG. "Randomized Algorithms for Optimizing Large Join Queries," In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Atlantic City, NJ, 1990, pp.312–321.

[KEMP90] A. KEMPER AND G. MOERKOTTE. "Access Support in Object Bases," In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Atlantic City, NJ, 1990, pp.364–386.

[KHOS89] S. KHOSHAFIAN AND P. VALDURIEZ. "A Parallel Container Model for Data Intensive Applications," In *Proc. Int. Workshop on Database Machines*, Deauville, France, June 1989, pp. 156–170.

[KHOS87] S. KHOSHAFIAN AND P. VALDURIEZ. "Sharing, Persistence and Object-Orientation, a database perspective," In *Proc. Int. Workshop on Database Programming Languages*, Roscoff, France, September 1987, pp.181–205.

[KIER90] G. KIERNAN, C. DE MAINDREVILLE, AND E. SIMON. "Making Deductive Database a Practical Technology: A Step Forward," In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Atlantic City, NJ, 1990, pp.237–246.

[KIM90] W. KIM, J.F. GARZA, N. BALLOU, AND D. WOELK. "Architecture of the ORION Next-Generation Database System," *IEEE Trans. Knowledge and Data Eng.* (March 1990), 2(1): 109–124.

[KIM89] W. KIM AND F.H. LOCHOVSKY (eds.). *Object-Oriented Concepts, Databases, and Applications*. New York, NY: ACM Press, 1989.

[KORT88] H. F. KORTH. "Optimization of Object-Retrieval Queries," In *Advances in Object Oriented Database Systems*, K. R. Dittrich (ed.), New York, NY: Springer-Verlag, 1988, pp. 352–357.

[KUIJ90] H. VAN KUIJK, F. PIJPERS, AND P. APERS. "Multilevel Query Optimization in Distributed Database Systems," In *Proc. Int. Conf. on Computing and Information*, Niagara Falls, Canada, May 1990, pp 295-300.

[LANZ90] R. LANZELOTTE AND P. VALDURIEZ. "Extending the Search Strategy in a Query Optimizer," Submitted for publication, 1990.

[LECL88] C. LÉCLUSE, P. RICHARD, AND F. VELEZ. "$O_2$: An Object-Oriented Data Model," In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Chicago, IL, 1988, pp.424–433.

[LISK88] B. LISKOV. Distributed Programming in Argus. *Commun. ACM* (March 1988), 31(3): 300–312.

[LISK86] B. LISKOV AND R. LADIN. Highly Available Distributed Services and Fault-Tolerant Distributed Garbage Collection. In *Proc. ACM Symp. Principles of Distributed Computing*, Vancouver, Canada, 1986, pp. 29–39.

[LITW88] W. LITWIN. From Database Systems to Multidatabase Systems: Why and How. In *Proc. British National Conference on Databases (BNCOD 6)*, Cambridge: Cambridge University Press, 1988, pp. 161–188.

[LITW87] W. LITWIN AND A. ABDELLATIF. An Overview of the Multidatabase Manipulation Language – MDL. *Proc. IEEE.* (May 1987), 75(5): 621–631.

[LYNC86] N. LYNCH AND M. MERRITT. *Introduction to the Theory of Nested Transactions.* Technical Report MIT/LCS/TR-367, Massachusetts Institute of Technology, Laboratory for Computer Science, July 1986.

[MAIE87] D. MAIER AND J. STEIN. "Development and Implementation of an Object-Oriented DBMS," In *Research Directions in Object-Oriented Programming*, B. Shriver and P. Wegner (eds.), Cambridge, MA: MIT Press, 1987, pp. 355–392.

[MANO90] F. MANOLA. "Object-Oriented Knowledge Bases - Parts I and II," *AI Expert* (March and April 1990), pp. 26–36 and pp. 46–57.

[MCCA89] D.R. MCCARTHY AND U. DAYAL. "The Architecture of an Active Data Base Management System," In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Portland, OR, 1989, pp.215–224.

[MINO82] T. MINOURA AND G. WIEDERHOLD. "Resilient Extended True-Copy Token Scheme for a Distributed Database System," *IEEE Trans. Software Eng.* (May 1982), SE-8(3): 173–189.

[MOSS85] E. MOSS. *Nested Transactions*. Cambridge, Mass.: MIT Press, 1985.

[MOTR81] A. MOTRO AND P. BUNEMAN. "Constructing Superviews," In *Proc. ACM SIGMOD Int. Conf. on Management of Data,* 1981, pp. 56–64.

[MURO85] S. MURO, T. IBARAKI, H. MIYAJIMA, AND T. HASEGAWA. "Evaluation of File Redundancy in Distributed Database Systems," *IEEE Trans. Software Eng.* (February 1985), SE-11(2): 199–205.

[NECH85] P.M. NECHES. "The Anatomy of a Database Computer," In *COMPCON Digest of Papers*, San Francisco, February 1985, pp. 252–254.

[OSBO88] S. L. OSBORN. "Identity, Equality and Query Optimization," In *Advances in Object Oriented Database Systems*, K. R. Dittrich (ed.), New York, NY: Springer-Verlag, 1988, pp. 346–351.

[ÖZSO87] Z.M. ÖZSOYOGLU AND N. ZHOU. "Distributed Query Processing in Broadcasting Local Area Networks," In *Proc. 20th Hawaii Int. Conf. on System Sciences*, Kailua-Kona, Hawaii, January 1987, pp. 419–429.

[ÖZSU91] M.T. ÖZSU AND P. VALDURIEZ. *Principles of Distributed Database Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1991.

[ÖZSU90a] M.T. ÖZSU AND D. MEECHAN. "Join Processing Heuristics in Relational Database Systems," *Information Systems* (1990), 15(4): 429-444.

[ÖZSU90b] M.T. ÖZSU AND D. MEECHAN. "Finding Heuristics for Processing Selection Queries in Relational Database Systems," *Information Systems* (1990), 15(3):359-373.

[RASH89] R. RASHID, R. BARON, A. FORIN, D. GOLUB, M. JONES, D. JULIN, D. ORR, AND R. SANZI. "Mach: A Foundation for Open Systems - A Position Paper," In *Proc. 2nd Workshop. on Workstation Operating Systems*; 1989, pp. 109-113.

[RENE87] R. VAN RENESSE, A.S. TANENBAUM, AND H. VAN STAVEREN. "Connecting RPC-Based Distributed Systems Using Wide-Area Networks," In *Proc. 7th Int. Conf. on Distributed Computing Systems*, May 1987.

[SCHE90a] H-J. SCHEK, H-B. PAUL, M.H. SCHOLL, AND G. WEIKUM. "The DASDBS Project: Objectives, Experiences, and Future Prospects," *IEEE Trans. Knowledge and Data Eng*. (March 1990), 2(1): 25–43.

[SCHE90b] P. SCHEURMANN AND C. YU (eds). "Report of the Workshop on Heterogeneous Database Systems," *IEEE Q. Bull. Database Eng*. (December 1990), 13(4): 3–11.

[SELL88] T.K. SELLIS. "Multiple query optimization," *ACM Trans. Database Syst*. (March 1988), 13(1): 23–52.

[SHAP90] M. SHAPIRO, O. GRUBER, AND D. PLAINFOSSE. *A Garbage Detection Protocol for Realistic Distributed Object-Support Systems*, Research Report No. 1320, INRIA, Rocquencourt, France, 1990.

[SHAW89] G. SHAW AND S.B. ZDONIK. "Object-Oriented Queries: Equivalence and Optimization," In *Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases,* Kyoto, Japan, 1989, pp. 264–278.

[SHET90] A. SHETH AND J. LARSON. "Federated Databases: Architectures and Integration," In [ELMA90], pp. 183–236.

[SILB90] A. SILBERSCHATZ, M. STONEBRAKER, AND J.D. ULLMAN (eds.). *Database Systems: Achievements and Opportunities – Report of the NSF Invitational Workshop on the Future of Database Systems Research*, Technical Report TR-90-22, Department of Computer Sciences, The University of Texas at Austin, Austin, TX, 1990.

[SIMO86] E. SIMON AND P. VALDURIEZ. Integrity Control in Distributed Database Systems. In *Proc. 19th Hawaii Int. Conf. on System Sciences*, Honolulu, January 1986, pp. 622–632.

[STON90a] M. STONEBRAKER. “Architecture of Future Data Base Systems,” *IEEE Q. Bull. Database Eng.* (December 1990), 13(4): 18–23.

[STON90b] M. STONEBRAKER, L. A. ROWE, B. LINDSAY, J. GRAY, M. CAREY, M. BRODIE, P. BERNSTEIN, AND D. BEECH. “Third-Generation Data Base System Manifesto.” *ACM SIGMOD Record* (September 1990), 19(3): 31–44.

[STON90c] M. STONEBRAKER, L.A. ROWE, AND M. HIROSHAMA. “The Implementation of POSTGRES,” *IEEE Trans. Knowledge and Data Eng.* (March 1990), 2(1): 125–142.

[STON89] M. STONEBRAKER. “Future Trends in Database Systems,” *IEEE Trans. Knowledge and Data Eng.* (March 1989), 1(1): 33–44.

[STON88] M. STONEBRAKER. *Readings in Database Systems*. San Mateo, Calif.: Morgan Kaufmann, 1988.

[STON81] M. STONEBRAKER. “Operating System Support for Database Management,” *Commun. ACM* (July 1981), 24(7): 412–418.

[STOY90] A. STOYENKO. “A General RPC for Model-Level Heterogeneous RPC Interoperability,” In *Proc. 2nd IEEE Symp. on Parallel and Distributed Processing*, Dallas, TX, December 1990, pp.668–675.

[STRA90] D.D. STRAUBE AND M.T. ÖZSU. “Queries and Query Processing in Object-Oriented Database Systems,” *ACM Trans. Inf. Syst.* (October 1990), 8(4): 387–430.

[TAND88] TANDEM PERFORMANCE GROUP. “A Benchmark of Non-Stop SQL on the Debit Credit Transaction,” In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Chicago, IL, 1988, pp. 337–341.

[TAND87] TANDEM PERFORMANCE GROUP. “NonStop SQL – A Distributed High-Performance, High-Availability Implementation of SQL,” In *Proc. Int. Workshop on High Performance Transaction Systems*, Asilomar, Calif., September 1987.

[TANE90] A.S. TANENBAUM. “Experiences with the Amoeba Distributed Operating System,” *Comm. ACM* (December 1990), 33(12): 46–63.

[TAYL87] R.W. TAYLOR. “Data Server Architectures: Experiences and Lessons,” In *Proc. CIPS (Canadian Information Processing Society) Edmonton '87 Conf.*, Edmonton, Alberta, Canada, 1987, pp. 334–342.

[THOM79] R.H. THOMAS. “A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases,” *ACM Trans. Database Syst.* (June 1979), 4(2): 180–209.

[VALD91] P. VALDURIEZ (ed.). *Data Management and Parallel Processing*, London, UK: Chapman and Hall, 1991 (to appear).

[VALD89] P. VALDURIEZ AND S. DANFORTH. “Query Optimization in Database Programming Languages,” In *Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases,* Kyoto, Japan, 1989, pp. 516–534.

[VALD88] P. VALDURIEZ AND S. KHOSHAFIAN. "Parallel Evaluation of the Transitive Closure of a Database Relation," *Int. J. Parallel Prog.* (February 1988), 12(1): 19–42.

[VALD84] P. VALDURIEZ AND G. GARDARIN. "Join and Semi-join Algorithms for a Multi Processor Database Machine," *ACM Trans. Database Syst.* (March 1984), 9(1): 133–161.

[WAH85] B.W. WAH AND Y.N. LIEN. "Design of Distributed Databases on Local Computer Systems," *IEEE Trans. Software Eng.* (July 1985), SE-11(7): 609–619.

[WEIH89] W.E. WEIHL. "The Impact of Recovery on Concurrency Control," In *Proc. 8th ACM SIGACT–SIGMOD–SIGART Symp. on Principles of Database Systems*, Philadelphia, PA, March 1989, pp. 259–269.

[WEIH88] W.E. WEIHL. "Commutativity-Based Concurrency Control for Abstract Data Types," *IEEE Trans. Comput.* (December 1988), 37(12): 1488–1505.

[WEIK86] G. WEIKUM. "Pros and Cons of Operating System Transactions for Data Base Systems," In *Proc. Fall Joint Computer Conf.*, Dallas, Tex., 1986, pp. 1219–1225.

[WOLF90] O. WOLFSON AND A. OZERI. "A New Paradigm for Parallel and Distributed Rule-Processing," In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Atlantic City, NJ, May 1990, pp.133–142.

[WOLS90] A. WOLSKI AND J. VEIJALAINEN. "2PC Agent Method: Achieving Serializability in Presence of Failures in a Heterogeneous Multidatabase," In *Proc. Int. Conf. on Databases, Parallel Architectures, and Their Applications*, Miami Beach, FL, 1990, pp. 321–330.

[WULF81] W.A. WULF, R. LEVIN, AND S.P. HARBISON. *HYDRA/C.mmp : An Experimental Computer System*. New York, N.Y.: McGraw Hill, 1981.

[YOSH85] M. YOSHIDA, K. MIZUMACHI, A. WAKINO, I. OYAKE, AND Y. MATSUSHITA. "Time and Cost Evaluation Schemes of Multiple Copies of Data in Distributed Database Systems," *IEEE Trans. Software Eng.* (September 1985), SE-11(9): 954–958.

[ZDON90] S. ZDONIK AND D. MAIER (eds.). *Readings in Object-Oriented Database Systems*, San Mateo, CA: Morgan Kaufmann, 1990.